Deep Learning with CUDA

Laboratory 01

Andrzej Świętek

Faculty of Physics and Applied Informatics

AGH

Introduction

In this lab session, our primary goal was to investigate the effects of different activation functions and batch sizes on the training process of neural network models. Specifically, we varied activation functions and batch sizes within the range of 8 to 512 to discern trends and understand the impact of altering these parameters.

Batch Size in Deep Learning

In deep learning, the batch size parameter refers to the number of training examples utilized in one iteration of the training process. This parameter plays a crucial role in determining the efficiency and effectiveness of model training.

A smaller batch size, such as 8 or 16, leads to more frequent updates to the model's parameters as each batch is processed, resulting in faster convergence but potentially at the expense of increased computational overhead due to frequent updates.

Conversely, a larger batch size, such as 256 or 512, allows for more stable updates and may lead to better utilization of computational resources, although it may require more epochs to converge and could suffer from overfitting if the batch size is too large relative to the dataset size.

Activation Function

The function responsible for activating neurons. It has two states that are being return based on the given threshold value : low – neuron not activated, high – neuron activated.

They introduce non-linearity to the network, allowing it to learn complex patterns and relationships within the data.

ReLU (Rectified Linear Unit):

ReLU is one of the most widely used activation functions due to its simplicity and effectiveness. It replaces all negative values with zero, effectively introducing non-linearity to the network.

Sigmoid:

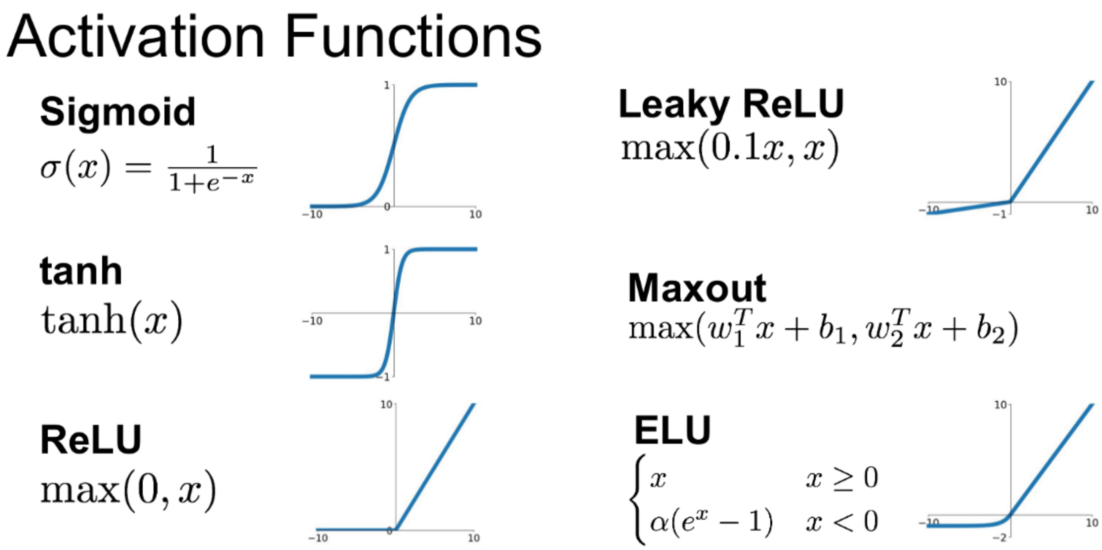
Sigmoid function squashes the output to a range between 0 and 1, making it suitable for binary classification tasks where the output needs to be interpreted as a probability.

Tanh (Hyperbolic Tangent):

Tanh function squashes the output to a range between -1 and 1, making it suitable for tasks where the output needs to be centered around zero.

Softmax:

Softmax function is often used as the activation function in the output layer of a neural network for multi-class classification tasks. It converts the raw scores into probabilities, with each output representing the probability of belonging to a particular class.

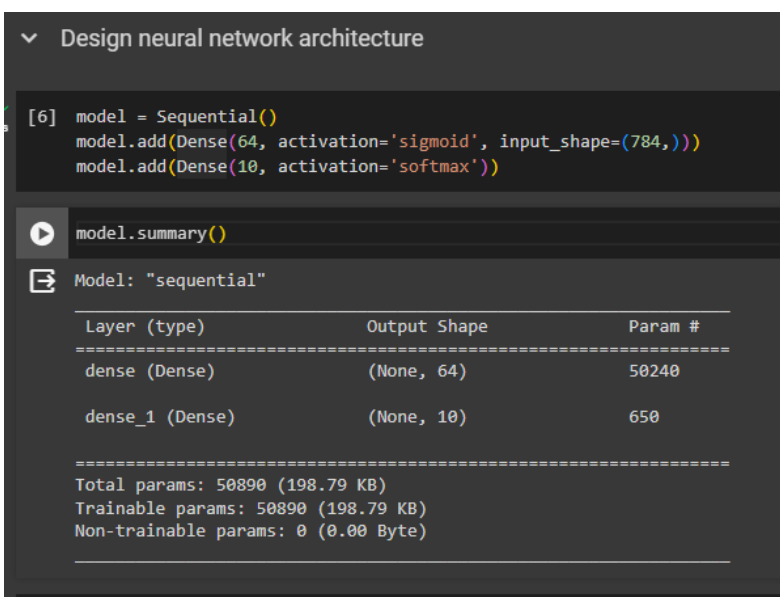


Picture 1: Activation Functions

Training model and its results

For all test cases I used the same number of epochs equal to 20. We consider the same shallow model with 2 Dense layer (connection all-to-all):

One input layer with sigmoid activation function and one output layer (softmax)



Batch Size 8

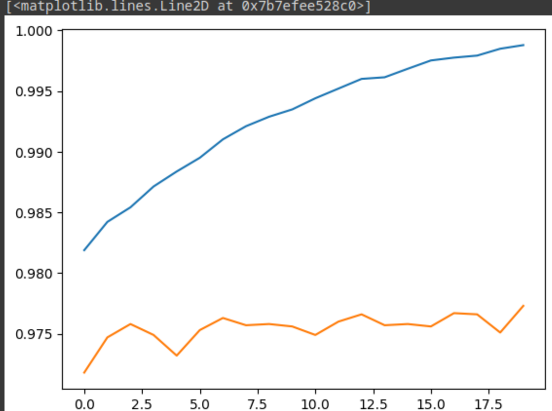
Train!

```
# evolution = model.fit(X_train, y_train, batch_size=128, epochs=20, verbose=1, validation_data=(X_valid, y_valid))
evolution = model.fit(X_train, y_train, batch_size=8, epochs=20, verbose=1, validation_data=(X_valid, y_valid))
type(evolution)
```

```
Epoch 1/20
7500/7500 [==============================] - 23s 3ms/step - loss: 0.0614 - accuracy: 0.9819 - val_loss: 0.0930 - val_accuracy: 0.9718
Epoch 2/20
7500/7500 [==============================] - 24s 3ms/step - loss: 0.0554 - accuracy: 0.9842 - val_loss: 0.0855 - val_accuracy: 0.9747
Epoch 3/20
7500/7500 [==============================] - 23s 3ms/step - loss: 0.0501 - accuracy: 0.9854 - val_loss: 0.0849 - val_accuracy: 0.9758
Epoch 4/20
7500/7500 [==============================] - 23s 3ms/step - loss: 0.0453 - accuracy: 0.9871 - val_loss: 0.0825 - val_accuracy: 0.9749
Epoch 5/20
7500/7500 [==============================] - 24s 3ms/step - loss: 0.0412 - accuracy: 0.9884 - val_loss: 0.0875 - val_accuracy: 0.9732
Epoch 6/20
7500/7500 [==============================] - 24s 3ms/step - loss: 0.0375 - accuracy: 0.9895 - val_loss: 0.0812 - val_accuracy: 0.9753
Epoch 7/20
7500/7500 [==============================] - 24s 3ms/step - loss: 0.0344 - accuracy: 0.9910 - val_loss: 0.0829 - val_accuracy: 0.9763
Epoch 8/20
7500/7500 [==============================] - 24s 3ms/step - loss: 0.0313 - accuracy: 0.9921 - val_loss: 0.0808 - val_accuracy: 0.9757
Epoch 9/20
7500/7500 [==============================] - 24s 3ms/step - loss: 0.0287 - accuracy: 0.9929 - val_loss: 0.0779 - val_accuracy: 0.9758
Epoch 10/20
7500/7500 [==============================] - 24s 3ms/step - loss: 0.0264 - accuracy: 0.9935 - val_loss: 0.0797 - val_accuracy: 0.9756
Epoch 11/20
7500/7500 [==============================] - 23s 3ms/step - loss: 0.0240 - accuracy: 0.9944 - val_loss: 0.0816 - val_accuracy: 0.9749
Epoch 12/20
7500/7500 [==============================] - 24s 3ms/step - loss: 0.0223 - accuracy: 0.9952 - val_loss: 0.0802 - val_accuracy: 0.9760
Epoch 13/20
7500/7500 [==============================] - 24s 3ms/step - loss: 0.0203 - accuracy: 0.9960 - val_loss: 0.0801 - val_accuracy: 0.9766
Epoch 14/20
7500/7500 [==============================] - 23s 3ms/step - loss: 0.0190 - accuracy: 0.9961 - val_loss: 0.0831 - val_accuracy: 0.9757
Epoch 15/20
7500/7500 [==============================] - 23s 3ms/step - loss: 0.0175 - accuracy: 0.9968 - val_loss: 0.0798 - val_accuracy: 0.9758
Epoch 16/20
7500/7500 [==============================] - 24s 3ms/step - loss: 0.0161 - accuracy: 0.9975 - val_loss: 0.0820 - val_accuracy: 0.9756
Epoch 17/20
7500/7500 [==============================] - 23s 3ms/step - loss: 0.0149 - accuracy: 0.9977 - val_loss: 0.0806 - val_accuracy: 0.9767
Epoch 18/20
7500/7500 [==============================] - 26s 3ms/step - loss: 0.0141 - accuracy: 0.9979 - val_loss: 0.0785 - val_accuracy: 0.9766
Epoch 19/20
7500/7500 [==============================] - 23s 3ms/step - loss: 0.0128 - accuracy: 0.9985 - val_loss: 0.0837 - val_accuracy: 0.9751
Epoch 20/20
7500/7500 [==============================] - 24s 3ms/step - loss: 0.0120 - accuracy: 0.9988 - val_loss: 0.0782 - val_accuracy: 0.9773
keras.src.callbacks.History
def __init__()

/usr/local/lib/python3.10/dist-packages/keras/src/callbacks.py
Callback that records events into a `History` object.

This callback is automatically applied to
every Keras model. The `History` object
gets returned by the `fit` method of models.
```

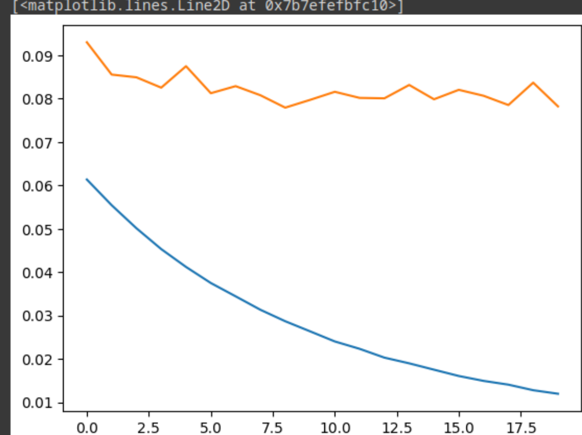
```
[58] print(evolution.history.keys())
plt.plot(evolution.history['accuracy'])
plt.plot(evolution.history['val_accuracy'])
```

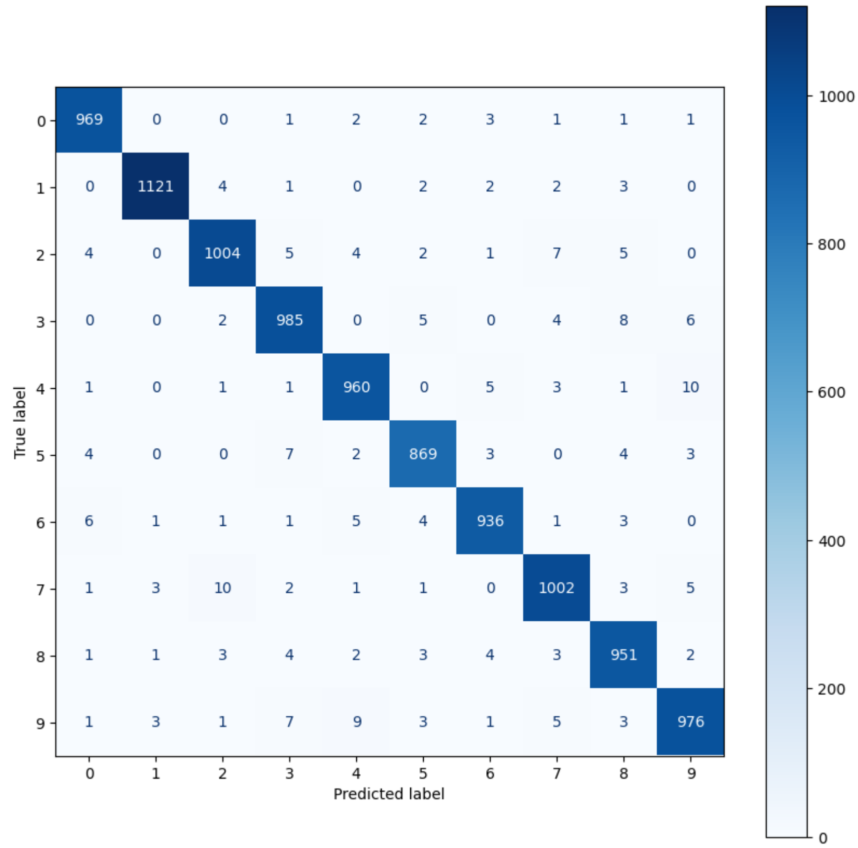
```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
[<matplotlib.lines.Line2D at 0x7b7efee528c0>]
```




```
plt.plot(evolution.history['loss'])
plt.plot(evolution.history['val_loss'])
```

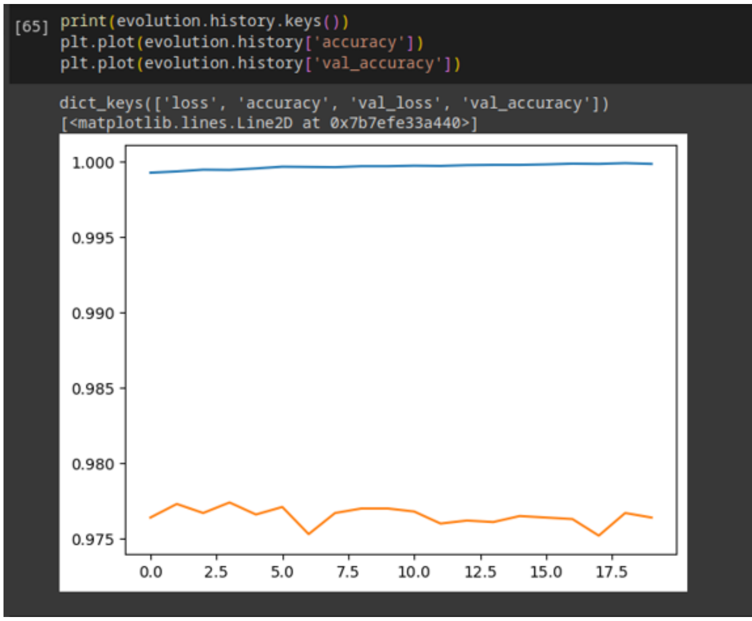
```
[<matplotlib.lines.Line2D at 0x7b7efefbfc10>]
```

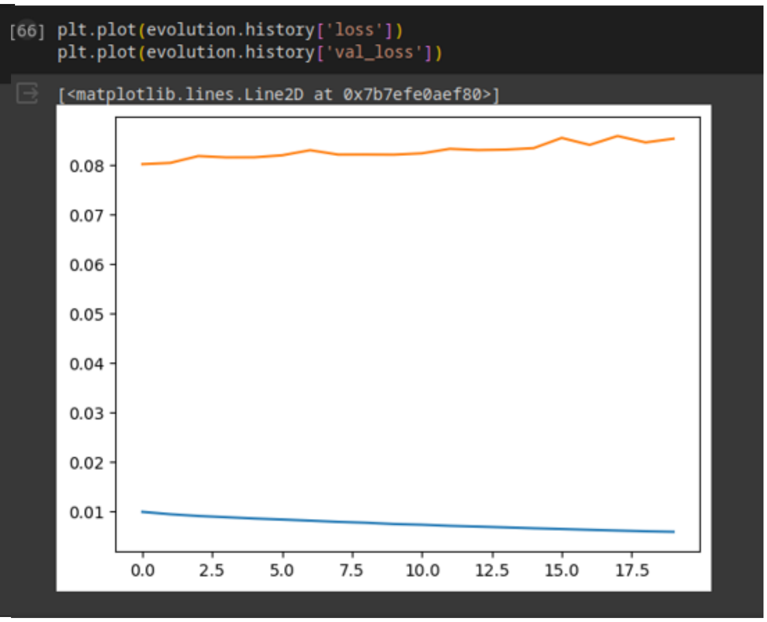


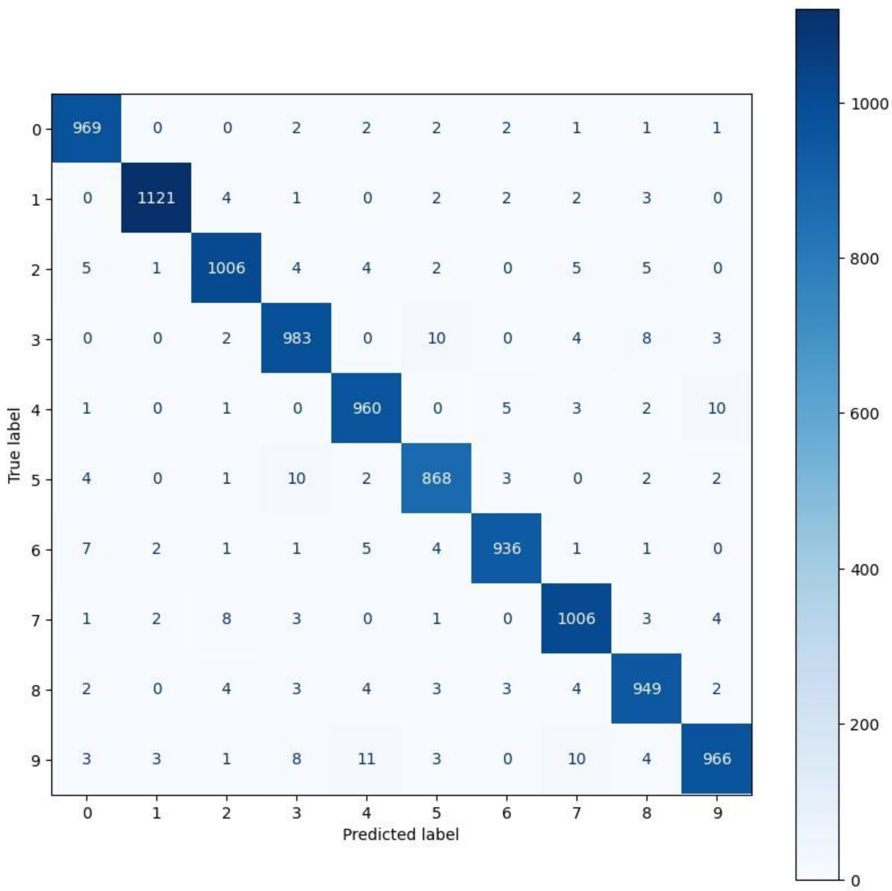


Batch Size 16

```
Train!

# evolution = model.fit(X_train, y_train, batch_size=128, epochs=20, verbose=1, validation_data=(X_valid, y_valid))
evolution = model.fit(X_train, y_train, batch_size=16, epochs=20, verbose=1, validation_data=(X_valid, y_valid))
type(evolution)

Epoch 1/20
3750/3750 [==============================] - 12s 3ms/step - loss: 0.0099 - accuracy: 0.9993 - val_loss: 0.0802 - val_accuracy: 0.9764
Epoch 2/20
3750/3750 [==============================] - 12s 3ms/step - loss: 0.0095 - accuracy: 0.9994 - val_loss: 0.0804 - val_accuracy: 0.9773
Epoch 3/20
3750/3750 [==============================] - 12s 3ms/step - loss: 0.0091 - accuracy: 0.9995 - val_loss: 0.0818 - val_accuracy: 0.9767
Epoch 4/20
3750/3750 [==============================] - 12s 3ms/step - loss: 0.0088 - accuracy: 0.9995 - val_loss: 0.0815 - val_accuracy: 0.9774
Epoch 5/20
3750/3750 [==============================] - 12s 3ms/step - loss: 0.0086 - accuracy: 0.9995 - val_loss: 0.0816 - val_accuracy: 0.9766
Epoch 6/20
3750/3750 [==============================] - 12s 3ms/step - loss: 0.0084 - accuracy: 0.9997 - val_loss: 0.0820 - val_accuracy: 0.9771
Epoch 7/20
3750/3750 [==============================] - 12s 3ms/step - loss: 0.0082 - accuracy: 0.9997 - val_loss: 0.0830 - val_accuracy: 0.9753
Epoch 8/20
3750/3750 [==============================] - 12s 3ms/step - loss: 0.0079 - accuracy: 0.9996 - val_loss: 0.0821 - val_accuracy: 0.9767
Epoch 9/20
3750/3750 [==============================] - 12s 3ms/step - loss: 0.0077 - accuracy: 0.9997 - val_loss: 0.0821 - val_accuracy: 0.9770
Epoch 10/20
3750/3750 [==============================] - 12s 3ms/step - loss: 0.0075 - accuracy: 0.9997 - val_loss: 0.0821 - val_accuracy: 0.9770
Epoch 11/20
3750/3750 [==============================] - 12s 3ms/step - loss: 0.0073 - accuracy: 0.9997 - val_loss: 0.0824 - val_accuracy: 0.9768
Epoch 12/20
3750/3750 [==============================] - 12s 3ms/step - loss: 0.0071 - accuracy: 0.9997 - val_loss: 0.0833 - val_accuracy: 0.9760
Epoch 13/20
3750/3750 [==============================] - 12s 3ms/step - loss: 0.0070 - accuracy: 0.9998 - val_loss: 0.0830 - val_accuracy: 0.9762
Epoch 14/20
3750/3750 [==============================] - 12s 3ms/step - loss: 0.0068 - accuracy: 0.9998 - val_loss: 0.0831 - val_accuracy: 0.9761
Epoch 15/20
3750/3750 [==============================] - 12s 3ms/step - loss: 0.0066 - accuracy: 0.9998 - val_loss: 0.0834 - val_accuracy: 0.9765
Epoch 16/20
3750/3750 [==============================] - 12s 3ms/step - loss: 0.0065 - accuracy: 0.9998 - val_loss: 0.0855 - val_accuracy: 0.9764
Epoch 17/20
3750/3750 [==============================] - 12s 3ms/step - loss: 0.0063 - accuracy: 0.9999 - val_loss: 0.0841 - val_accuracy: 0.9763
Epoch 18/20
3750/3750 [==============================] - 12s 3ms/step - loss: 0.0062 - accuracy: 0.9998 - val_loss: 0.0859 - val_accuracy: 0.9752
Epoch 19/20
3750/3750 [==============================] - 12s 3ms/step - loss: 0.0060 - accuracy: 0.9999 - val_loss: 0.0846 - val_accuracy: 0.9767
Epoch 20/20
3750/3750 [==============================] - 12s 3ms/step - loss: 0.0059 - accuracy: 0.9998 - val_loss: 0.0853 - val_accuracy: 0.9764
keras.src.callbacks.History
```


[65] print(evolution.history.keys())
plt.plot(evolution.history['accuracy'])
plt.plot(evolution.history['val_accuracy'])
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
[<matplotlib.lines.Line2D at 0x7b7efe33a440>]


[66] plt.plot(evolution.history['loss'])
plt.plot(evolution.history['val_loss'])
[<matplotlib.lines.Line2D at 0x7b7efe0aef80>]


True label
Predicted label

Batch Size 32

Train!

```
# evolution = model.fit(X_train, y_train, batch_size=128, epochs=20, verbose=1, validation_data=(X_valid, y_valid))
evolution = model.fit(X_train, y_train, batch_size=32, epochs=20, verbose=1, validation_data=(X_valid, y_valid))
type(evolution)
```

```
Epoch 1/20
1875/1875 [==============================] - 6s 3ms/step - loss: 0.0055 - accuracy: 0.9999 - val_loss: 0.0845 - val_accuracy: 0.9765
Epoch 2/20
1875/1875 [==============================] - 6s 3ms/step - loss: 0.0055 - accuracy: 0.9999 - val_loss: 0.0846 - val_accuracy: 0.9761
Epoch 3/20
1875/1875 [==============================] - 6s 3ms/step - loss: 0.0054 - accuracy: 0.9999 - val_loss: 0.0853 - val_accuracy: 0.9764
Epoch 4/20
1875/1875 [==============================] - 7s 3ms/step - loss: 0.0054 - accuracy: 0.9999 - val_loss: 0.0851 - val_accuracy: 0.9763
Epoch 5/20
1875/1875 [==============================] - 6s 3ms/step - loss: 0.0053 - accuracy: 0.9999 - val_loss: 0.0854 - val_accuracy: 0.9761
Epoch 6/20
1875/1875 [==============================] - 7s 4ms/step - loss: 0.0053 - accuracy: 0.9999 - val_loss: 0.0851 - val_accuracy: 0.9766
Epoch 7/20
1875/1875 [==============================] - 6s 3ms/step - loss: 0.0052 - accuracy: 0.9999 - val_loss: 0.0853 - val_accuracy: 0.9763
Epoch 8/20
1875/1875 [==============================] - 7s 4ms/step - loss: 0.0052 - accuracy: 0.9999 - val_loss: 0.0852 - val_accuracy: 0.9765
Epoch 9/20
1875/1875 [==============================] - 6s 3ms/step - loss: 0.0051 - accuracy: 0.9999 - val_loss: 0.0857 - val_accuracy: 0.9761
Epoch 10/20
1875/1875 [==============================] - 6s 3ms/step - loss: 0.0051 - accuracy: 0.9999 - val_loss: 0.0854 - val_accuracy: 0.9763
Epoch 11/20
1875/1875 [==============================] - 6s 3ms/step - loss: 0.0050 - accuracy: 0.9999 - val_loss: 0.0855 - val_accuracy: 0.9764
Epoch 12/20
1875/1875 [==============================] - 7s 4ms/step - loss: 0.0050 - accuracy: 1.0000 - val_loss: 0.0863 - val_accuracy: 0.9757
Epoch 13/20
1875/1875 [==============================] - 6s 3ms/step - loss: 0.0049 - accuracy: 0.9999 - val_loss: 0.0858 - val_accuracy: 0.9764
Epoch 14/20
1875/1875 [==============================] - 7s 4ms/step - loss: 0.0049 - accuracy: 1.0000 - val_loss: 0.0862 - val_accuracy: 0.9757
Epoch 15/20
1875/1875 [==============================] - 5s 3ms/step - loss: 0.0048 - accuracy: 1.0000 - val_loss: 0.0862 - val_accuracy: 0.9759
Epoch 16/20
1875/1875 [==============================] - 6s 3ms/step - loss: 0.0048 - accuracy: 1.0000 - val_loss: 0.0867 - val_accuracy: 0.9761
Epoch 17/20
1875/1875 [==============================] - 6s 3ms/step - loss: 0.0047 - accuracy: 1.0000 - val_loss: 0.0857 - val_accuracy: 0.9766
Epoch 18/20
1875/1875 [==============================] - 7s 4ms/step - loss: 0.0047 - accuracy: 0.9999 - val_loss: 0.0862 - val_accuracy: 0.9760
Epoch 19/20
1875/1875 [==============================] - 6s 3ms/step - loss: 0.0047 - accuracy: 1.0000 - val_loss: 0.0866 - val_accuracy: 0.9759
Epoch 20/20
1875/1875 [==============================] - 6s 3ms/step - loss: 0.0046 - accuracy: 1.0000 - val_loss: 0.0865 - val_accuracy: 0.9758
```

```
keras.src.callbacks.History
def __init__()

/usr/local/lib/python3.10/dist-packages/keras/src/callbacks.py
Callback that records events into a `History` object.

This callback is automatically applied to
every Keras model. The `History` object
gets returned by the `fit` method of models.
```

+ Code + Text

Evaluating model performance

```
[74] print(X_valid.shape)
     print(y_valid.shape)
     # evaluate the model using the entire validation data set
     model.evaluate(X_valid, y_valid)
     #from sklearn.metrics import ConfusionMatrixDisplay
     #from sklearn.metrics import confusion_matrix
     #labels = [str(digit) for digit in range(10)]
     #y_pred = np.array(y_predicted)
     #full_cm = confusion_matrix(y_valid, y_pred)
     #y_pred.shape
```

```
(10000, 784)
(10000, 10)
313/313 [==============================] - 1s 2ms/step - loss: 0.0865 - accuracy: 0.9758
[0.08648288249969482, 0.9757999777793884]
```

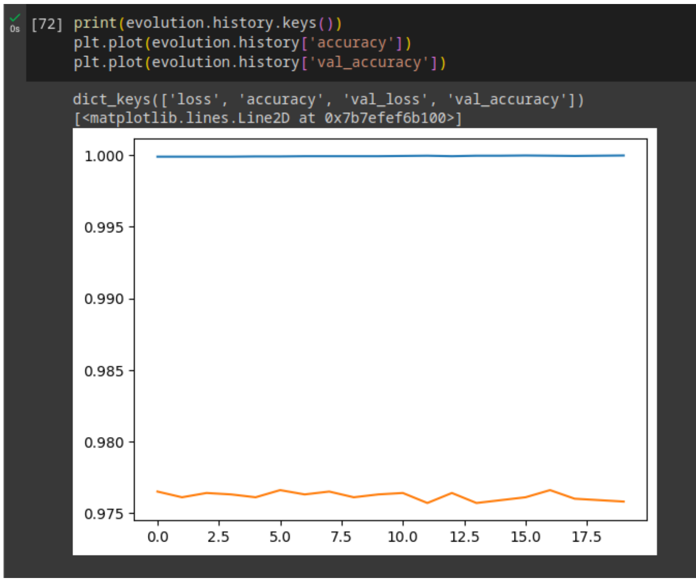
Performing inference

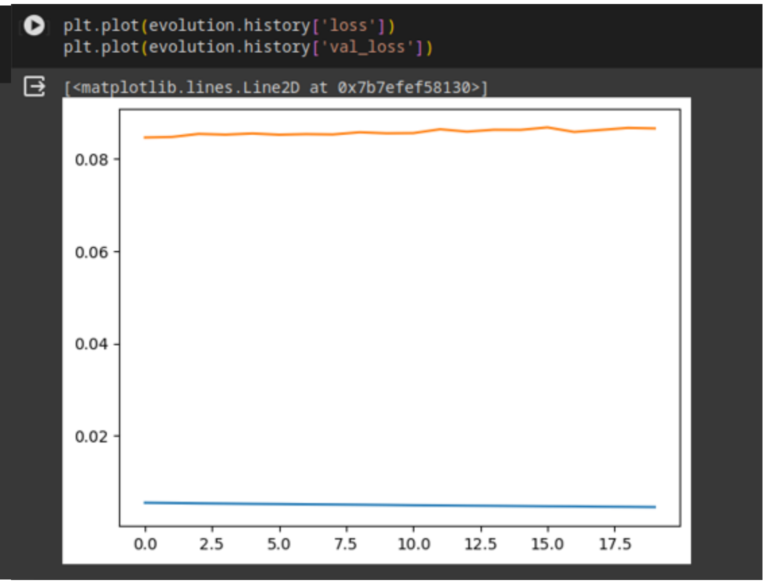
```
[75] print(X_valid[0].shape)
     valid_0 = X_valid[0].reshape(1, 784)
     X_valid.shape
```

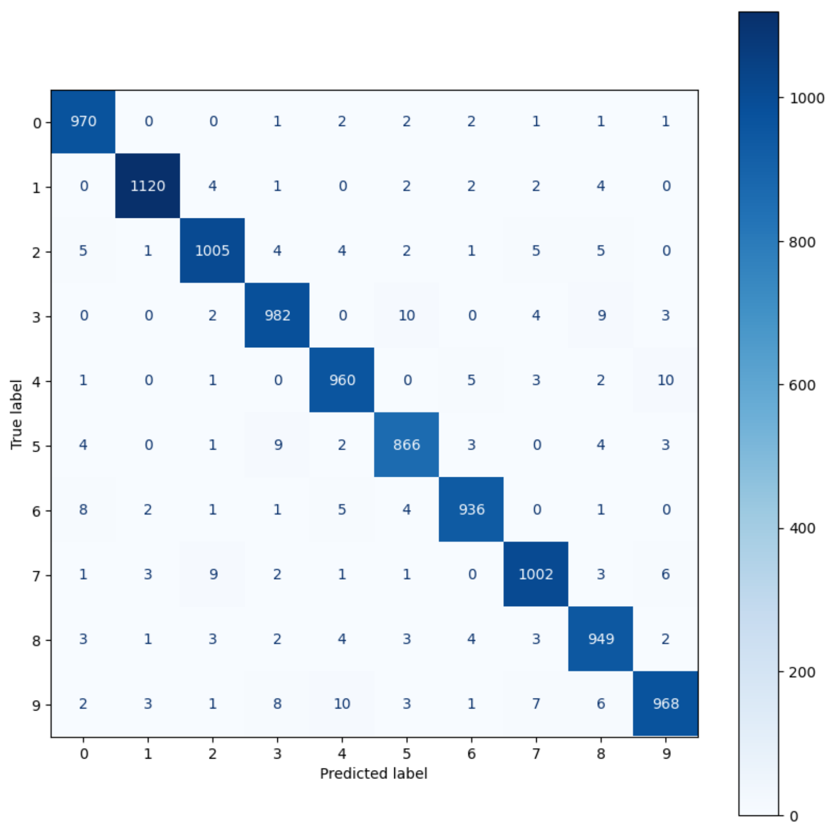
```
(784,)
(10000, 784)
```

```
# make a prediction with a single and entire data set
class_predict = model.predict(valid_0)
print(class_predict)
class_predicts = model.predict(X_valid)
print(class_predicts.shape)
```

```
1/1 [==============================] - 0s 20ms/step
[[3.8517970e-07 1.6582818e-10 2.5737833e-07 3.2427033e-07 1.6719748e-12
  4.2088924e-10 7.2464339e-14 9.9999833e-01 8.9959364e-09 6.5218802e-07]]
313/313 [==============================] - 0s 1ms/step
(10000, 10)
```


[72] print(evolution.history.keys())
plt.plot(evolution.history['accuracy'])
plt.plot(evolution.history['val_accuracy'])
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
[<matplotlib.lines.Line2D at 0x7b7efef6b100>]
1.000
0.995
0.990
0.985
0.980
0.975
0.0 2.5 5.0 7.5 10.0 12.5 15.0 17.5


plt.plot(evolution.history['loss'])
plt.plot(evolution.history['val_loss'])
[<matplotlib.lines.Line2D at 0x7b7efef58130>]
0.08
0.06
0.04
0.02
0.0 2.5 5.0 7.5 10.0 12.5 15.0 17.5


True label
Predicted label
0: 970 0 0 1 2 2 2 1 1 1
1: 0 1120 4 1 0 2 2 2 4 0
2: 5 1 1005 4 4 2 1 5 5 0
3: 0 0 2 982 0 10 0 4 9 3
4: 1 0 1 0 960 0 5 3 2 10
5: 4 0 1 9 2 866 3 0 4 3
6: 8 2 1 1 5 4 936 0 1 0
7: 1 3 9 2 1 1 0 1002 3 6
8: 3 1 3 2 4 3 4 3 949 2
9: 2 3 1 8 10 3 1 7 6 968
0 1 2 3 4 5 6 7 8 9
1000
800
600
400
200
0

Batch Size 64

Train!

```
# evolution = model.fit(X_train, y_train, batch_size=128, epochs=20, verbose=1, validation_data=(X_valid, y_valid))
evolution = model.fit(X_train, y_train, batch_size=64, epochs=20, verbose=1, validation_data=(X_valid, y_valid))
type(evolution)
```

```
Epoch 1/20
938/938 [==============================] - 3s 3ms/step - loss: 0.0045 - accuracy: 1.0000 - val_loss: 0.0866 - val_accuracy: 0.9759
Epoch 2/20
938/938 [==============================] - 3s 3ms/step - loss: 0.0045 - accuracy: 1.0000 - val_loss: 0.0862 - val_accuracy: 0.9761
Epoch 3/20
938/938 [==============================] - 4s 4ms/step - loss: 0.0045 - accuracy: 1.0000 - val_loss: 0.0865 - val_accuracy: 0.9760
Epoch 4/20
938/938 [==============================] - 3s 3ms/step - loss: 0.0044 - accuracy: 1.0000 - val_loss: 0.0866 - val_accuracy: 0.9761
Epoch 5/20
938/938 [==============================] - 3s 3ms/step - loss: 0.0044 - accuracy: 1.0000 - val_loss: 0.0867 - val_accuracy: 0.9758
Epoch 6/20
938/938 [==============================] - 3s 3ms/step - loss: 0.0044 - accuracy: 1.0000 - val_loss: 0.0868 - val_accuracy: 0.9759
Epoch 7/20
938/938 [==============================] - 4s 4ms/step - loss: 0.0044 - accuracy: 1.0000 - val_loss: 0.0867 - val_accuracy: 0.9761
Epoch 8/20
938/938 [==============================] - 3s 3ms/step - loss: 0.0044 - accuracy: 1.0000 - val_loss: 0.0865 - val_accuracy: 0.9761
Epoch 9/20
938/938 [==============================] - 3s 3ms/step - loss: 0.0043 - accuracy: 1.0000 - val_loss: 0.0866 - val_accuracy: 0.9760
Epoch 10/20
938/938 [==============================] - 3s 3ms/step - loss: 0.0043 - accuracy: 1.0000 - val_loss: 0.0867 - val_accuracy: 0.9760
Epoch 11/20
938/938 [==============================] - 3s 4ms/step - loss: 0.0043 - accuracy: 1.0000 - val_loss: 0.0869 - val_accuracy: 0.9760
Epoch 12/20
938/938 [==============================] - 3s 3ms/step - loss: 0.0043 - accuracy: 1.0000 - val_loss: 0.0874 - val_accuracy: 0.9759
Epoch 13/20
938/938 [==============================] - 3s 3ms/step - loss: 0.0043 - accuracy: 1.0000 - val_loss: 0.0872 - val_accuracy: 0.9758
Epoch 14/20
938/938 [==============================] - 3s 3ms/step - loss: 0.0042 - accuracy: 1.0000 - val_loss: 0.0871 - val_accuracy: 0.9760
Epoch 15/20
938/938 [==============================] - 3s 4ms/step - loss: 0.0042 - accuracy: 1.0000 - val_loss: 0.0870 - val_accuracy: 0.9759
Epoch 16/20
938/938 [==============================] - 3s 3ms/step - loss: 0.0042 - accuracy: 1.0000 - val_loss: 0.0871 - val_accuracy: 0.9760
Epoch 17/20
938/938 [==============================] - 3s 3ms/step - loss: 0.0042 - accuracy: 1.0000 - val_loss: 0.0873 - val_accuracy: 0.9759
Epoch 18/20
938/938 [==============================] - 3s 3ms/step - loss: 0.0042 - accuracy: 1.0000 - val_loss: 0.0871 - val_accuracy: 0.9759
Epoch 19/20
938/938 [==============================] - 3s 3ms/step - loss: 0.0042 - accuracy: 1.0000 - val_loss: 0.0874 - val_accuracy: 0.9758
Epoch 20/20
938/938 [==============================] - 3s 3ms/step - loss: 0.0042 - accuracy: 1.0000 - val_loss: 0.0873 - val_accuracy: 0.9759
```

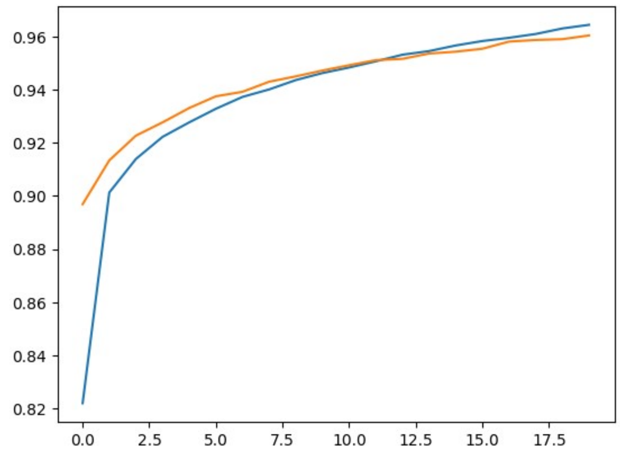
+ Code + Text

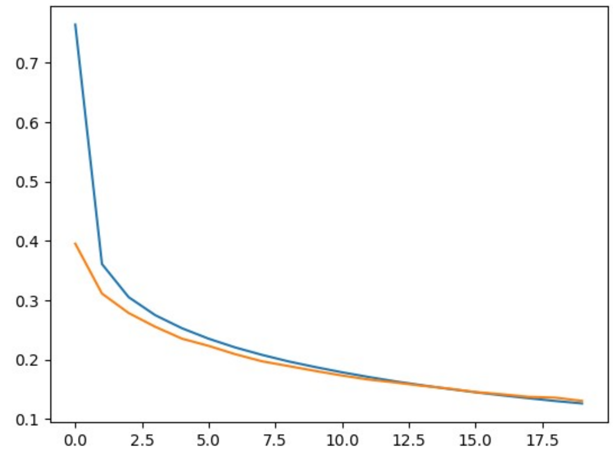
Evaluating model performance

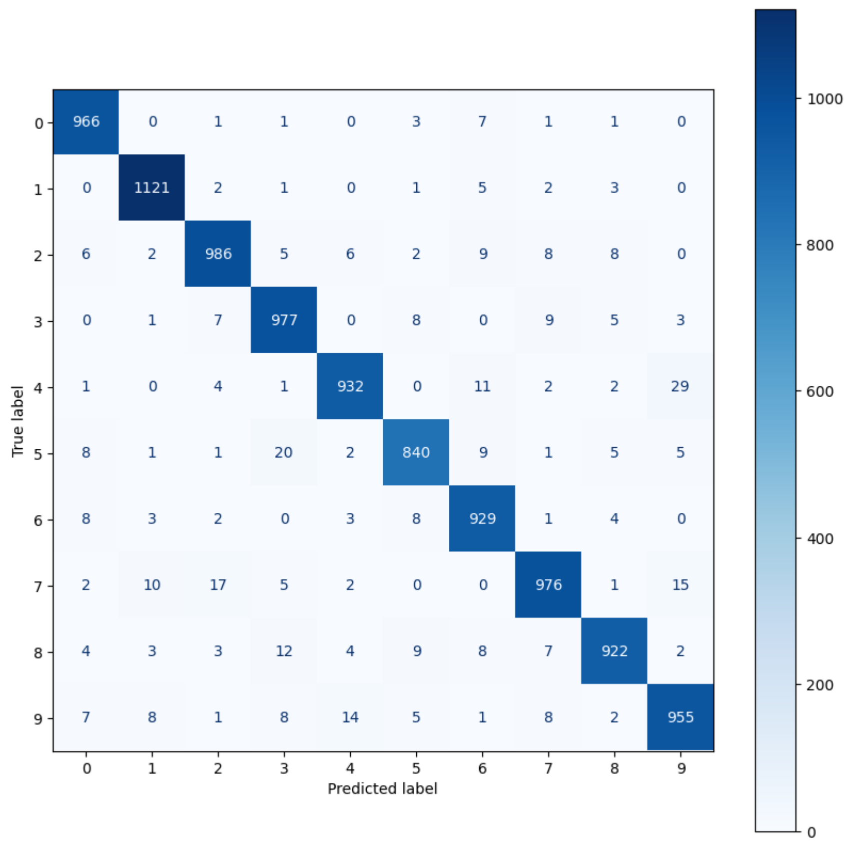
[81]
```
print(X_valid.shape)
print(y_valid.shape)
# evaluate the model using the entire validation data set
model.evaluate(X_valid, y_valid)
#from sklearn.metrics import ConfusionMatrixDisplay
#from sklearn.metrics import confusion_matrix
#labels = [str(digit) for digit in range(10)]
#y_pred = np.array(y_predicted)
#full_cm = confusion_matrix(y_valid, y_pred)
#y_pred.shape
```

```
(10000, 784)
(10000, 10)
313/313 [==============================] - 1s 2ms/step - loss: 0.0873 - accuracy: 0.9759
[0.08730589598417282, 0.9758999943733215]
```

Performing inference






True label
Predicted label

Batch Size 128

```
evolution = model.fit(X_train, y_train, batch_size=128, epochs=20, verbose=1, validation_data=(X_valid, y_valid))
# evolution = model.fit(X_train, y_train, batch_size=64, epochs=20, verbose=1, validation_data=(X_valid, y_valid))
type(evolution)
```

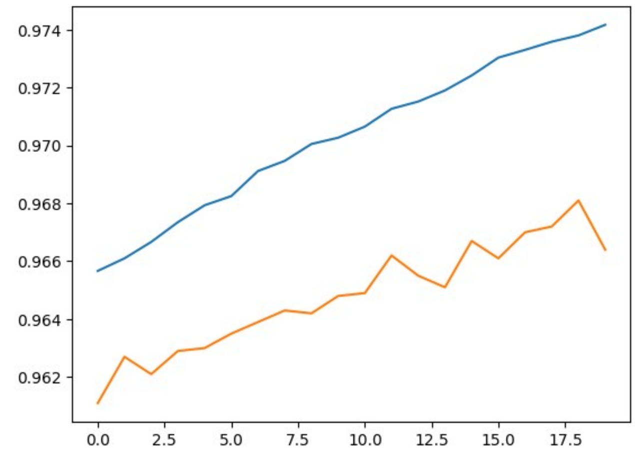
```
Epoch 1/20
469/469 [==============================] - 2s 4ms/step - loss: 0.1215 - accuracy: 0.9657 - val_loss: 0.1288 - val_accuracy: 0.9611
Epoch 2/20
469/469 [==============================] - 2s 4ms/step - loss: 0.1195 - accuracy: 0.9661 - val_loss: 0.1265 - val_accuracy: 0.9627
Epoch 3/20
469/469 [==============================] - 2s 3ms/step - loss: 0.1177 - accuracy: 0.9667 - val_loss: 0.1247 - val_accuracy: 0.9621
Epoch 4/20
469/469 [==============================] - 2s 4ms/step - loss: 0.1159 - accuracy: 0.9674 - val_loss: 0.1236 - val_accuracy: 0.9629
Epoch 5/20
469/469 [==============================] - 2s 3ms/step - loss: 0.1141 - accuracy: 0.9679 - val_loss: 0.1224 - val_accuracy: 0.9630
Epoch 6/20
469/469 [==============================] - 2s 3ms/step - loss: 0.1125 - accuracy: 0.9682 - val_loss: 0.1209 - val_accuracy: 0.9635
Epoch 7/20
469/469 [==============================] - 2s 3ms/step - loss: 0.1107 - accuracy: 0.9691 - val_loss: 0.1195 - val_accuracy: 0.9639
Epoch 8/20
469/469 [==============================] - 2s 3ms/step - loss: 0.1091 - accuracy: 0.9695 - val_loss: 0.1181 - val_accuracy: 0.9643
Epoch 9/20
469/469 [==============================] - 2s 5ms/step - loss: 0.1076 - accuracy: 0.9700 - val_loss: 0.1169 - val_accuracy: 0.9642
Epoch 10/20
469/469 [==============================] - 2s 4ms/step - loss: 0.1061 - accuracy: 0.9703 - val_loss: 0.1156 - val_accuracy: 0.9648
Epoch 11/20
469/469 [==============================] - 2s 4ms/step - loss: 0.1045 - accuracy: 0.9707 - val_loss: 0.1149 - val_accuracy: 0.9649
Epoch 12/20
469/469 [==============================] - 2s 3ms/step - loss: 0.1032 - accuracy: 0.9713 - val_loss: 0.1134 - val_accuracy: 0.9662
Epoch 13/20
469/469 [==============================] - 2s 4ms/step - loss: 0.1017 - accuracy: 0.9715 - val_loss: 0.1125 - val_accuracy: 0.9655
Epoch 14/20
469/469 [==============================] - 2s 4ms/step - loss: 0.1003 - accuracy: 0.9719 - val_loss: 0.1130 - val_accuracy: 0.9651
Epoch 15/20
469/469 [==============================] - 2s 4ms/step - loss: 0.0991 - accuracy: 0.9724 - val_loss: 0.1103 - val_accuracy: 0.9667
Epoch 16/20
469/469 [==============================] - 2s 4ms/step - loss: 0.0977 - accuracy: 0.9730 - val_loss: 0.1099 - val_accuracy: 0.9661
Epoch 17/20
469/469 [==============================] - 2s 3ms/step - loss: 0.0965 - accuracy: 0.9733 - val_loss: 0.1087 - val_accuracy: 0.9670
Epoch 18/20
469/469 [==============================] - 2s 4ms/step - loss: 0.0953 - accuracy: 0.9736 - val_loss: 0.1084 - val_accuracy: 0.9672
Epoch 19/20
469/469 [==============================] - 2s 4ms/step - loss: 0.0941 - accuracy: 0.9738 - val_loss: 0.1069 - val_accuracy: 0.9681
Epoch 20/20
469/469 [==============================] - 1s 3ms/step - loss: 0.0929 - accuracy: 0.9742 - val_loss: 0.1065 - val_accuracy: 0.9664
```

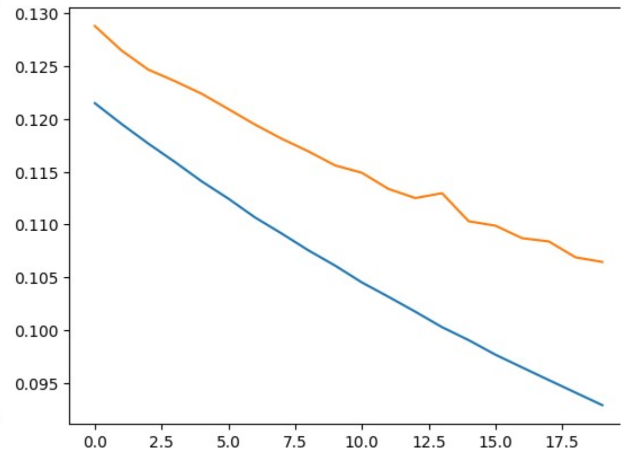
Evaluating model performance

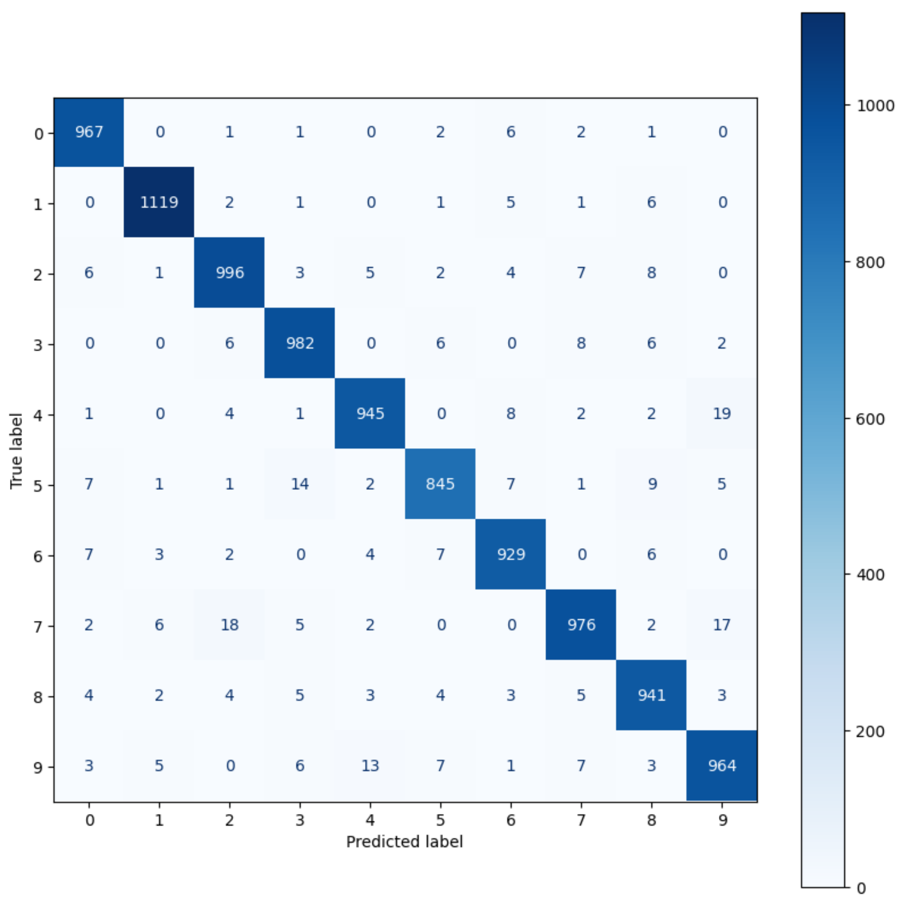
0s

```
print(X_valid.shape)
print(y_valid.shape)
# evaluate the model using the entire validation data set
model.evaluate(X_valid, y_valid)
#from sklearn.metrics import ConfusionMatrixDisplay
#from sklearn.metrics import confusion_matrix
#labels = [str(digit) for digit in range(10)]
#y_pred = np.array(y_predicted)
#full_cm = confusion_matrix(y_valid, y_pred)
#y_pred.shape
```

```
(10000, 784)
(10000, 10)
313/313 [==============================] - 1s 2ms/step - loss: 0.1065 - accuracy: 0.9664
[0.10646267980337143, 0.9664000272750854]
```


0.974
0.972
0.970
0.968
0.966
0.964
0.962
0.0 2.5 5.0 7.5 10.0 12.5 15.0 17.5


0.130
0.125
0.120
0.115
0.110
0.105
0.100
0.095
0.0 2.5 5.0 7.5 10.0 12.5 15.0 17.5


True label
Predicted label
967 0 1 1 0 2 6 2 1 0
0 1119 2 1 0 1 5 1 6 0
6 1 996 3 5 2 4 7 8 0
0 0 6 982 0 6 0 8 6 2
1 0 4 1 945 0 8 2 2 19
7 1 1 14 2 845 7 1 9 5
7 3 2 0 4 7 929 0 6 0
2 6 18 5 2 0 0 976 2 17
4 2 4 5 3 4 3 5 941 3
3 5 0 6 13 7 1 7 3 964
1000
800
600
400
200
0

Batch Size 256

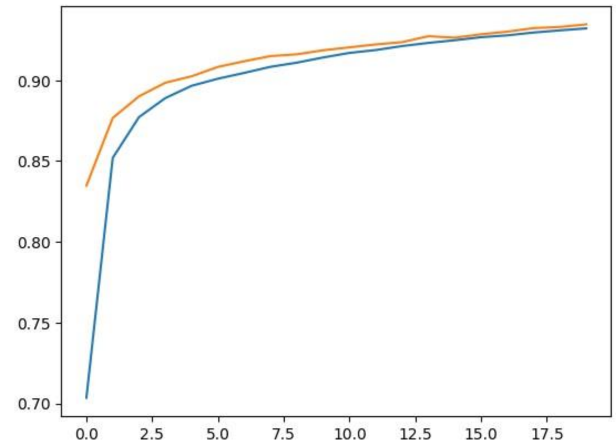
```
evolution = model.fit(X_train, y_train, batch_size=256, epochs=20, verbose=1, validation_data=(X_valid, y_valid))
type(evolution)
```

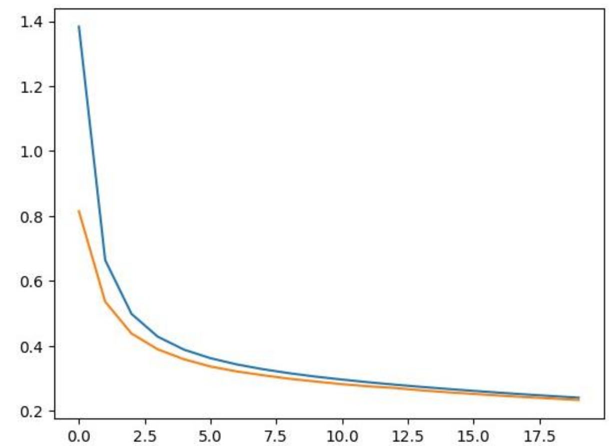
```
Epoch 1/20
235/235 [==============================] - 2s 5ms/step - loss: 1.3832 - accuracy: 0.7035 - val_loss: 0.8148 - val_accuracy: 0.8348
Epoch 2/20
235/235 [==============================] - 1s 5ms/step - loss: 0.6642 - accuracy: 0.8519 - val_loss: 0.5361 - val_accuracy: 0.8767
Epoch 3/20
235/235 [==============================] - 1s 3ms/step - loss: 0.4986 - accuracy: 0.8773 - val_loss: 0.4382 - val_accuracy: 0.8901
Epoch 4/20
235/235 [==============================] - 1s 4ms/step - loss: 0.4283 - accuracy: 0.8890 - val_loss: 0.3893 - val_accuracy: 0.8985
Epoch 5/20
235/235 [==============================] - 1s 4ms/step - loss: 0.3884 - accuracy: 0.8966 - val_loss: 0.3588 - val_accuracy: 0.9025
Epoch 6/20
235/235 [==============================] - 1s 4ms/step - loss: 0.3622 - accuracy: 0.9010 - val_loss: 0.3366 - val_accuracy: 0.9083
Epoch 7/20
235/235 [==============================] - 1s 4ms/step - loss: 0.3431 - accuracy: 0.9046 - val_loss: 0.3215 - val_accuracy: 0.9118
Epoch 8/20
235/235 [==============================] - 1s 4ms/step - loss: 0.3282 - accuracy: 0.9084 - val_loss: 0.3097 - val_accuracy: 0.9150
Epoch 9/20
235/235 [==============================] - 1s 4ms/step - loss: 0.3159 - accuracy: 0.9109 - val_loss: 0.2989 - val_accuracy: 0.9161
Epoch 10/20
235/235 [==============================] - 1s 4ms/step - loss: 0.3054 - accuracy: 0.9141 - val_loss: 0.2902 - val_accuracy: 0.9186
Epoch 11/20
235/235 [==============================] - 1s 4ms/step - loss: 0.2964 - accuracy: 0.9170 - val_loss: 0.2822 - val_accuracy: 0.9204
Epoch 12/20
235/235 [==============================] - 1s 4ms/step - loss: 0.2881 - accuracy: 0.9187 - val_loss: 0.2757 - val_accuracy: 0.9222
Epoch 13/20
235/235 [==============================] - 1s 4ms/step - loss: 0.2808 - accuracy: 0.9212 - val_loss: 0.2705 - val_accuracy: 0.9236
Epoch 14/20
235/235 [==============================] - 1s 5ms/step - loss: 0.2739 - accuracy: 0.9232 - val_loss: 0.2633 - val_accuracy: 0.9273
Epoch 15/20
235/235 [==============================] - 1s 5ms/step - loss: 0.2675 - accuracy: 0.9248 - val_loss: 0.2573 - val_accuracy: 0.9264
Epoch 16/20
235/235 [==============================] - 1s 4ms/step - loss: 0.2616 - accuracy: 0.9267 - val_loss: 0.2522 - val_accuracy: 0.9285
Epoch 17/20
235/235 [==============================] - 1s 4ms/step - loss: 0.2558 - accuracy: 0.9278 - val_loss: 0.2470 - val_accuracy: 0.9301
Epoch 18/20
235/235 [==============================] - 1s 4ms/step - loss: 0.2505 - accuracy: 0.9296 - val_loss: 0.2423 - val_accuracy: 0.9323
Epoch 19/20
235/235 [==============================] - 1s 4ms/step - loss: 0.2455 - accuracy: 0.9309 - val_loss: 0.2380 - val_accuracy: 0.9330
Epoch 20/20
235/235 [==============================] - 1s 4ms/step - loss: 0.2405 - accuracy: 0.9321 - val_loss: 0.2335 - val_accuracy: 0.9346
```

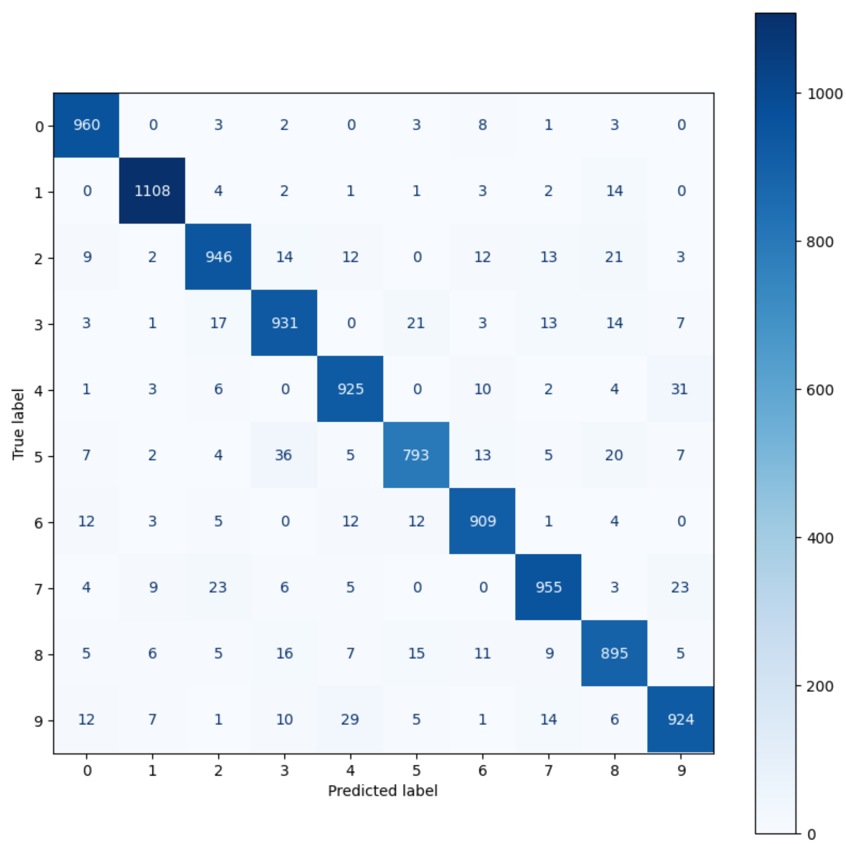
Evaluating model performance

```
[51] print(X_valid.shape)
     print(y_valid.shape)
     # evaluate the model using the entire validation data set
     model.evaluate(X_valid, y_valid)
     #from sklearn.metrics import ConfusionMatrixDisplay
     #from sklearn.metrics import confusion_matrix
     #labels = [str(digit) for digit in range(10)]
     #y_pred = np.array(y_predicted)
     #full_cm = confusion_matrix(y_valid, y_pred)
     #y_pred.shape
```

```
(10000, 784)
(10000, 10)
313/313 [==============================] - 1s 2ms/step - loss: 0.2335 - accuracy: 0.9346
[0.23351271450519562, 0.9345999956130981]
```






True label
Predicted label

Batch Size 512

```
evolution = model.fit(X_train, y_train, batch_size=512, epochs=20, verbose=1, validation_data=(X_valid, y_valid))
type(evolution)
```

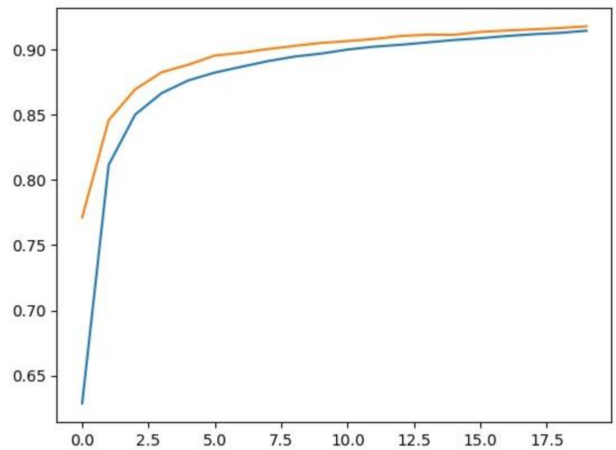
```
Epoch 1/20
118/118 [==============================] - 1s 6ms/step - loss: 1.7449 - accuracy: 0.6285 - val_loss: 1.2597 - val_accuracy: 0.7713
Epoch 2/20
118/118 [==============================] - 0s 4ms/step - loss: 1.0152 - accuracy: 0.8116 - val_loss: 0.8069 - val_accuracy: 0.8459
Epoch 3/20
118/118 [==============================] - 0s 4ms/step - loss: 0.7260 - accuracy: 0.8503 - val_loss: 0.6266 - val_accuracy: 0.8696
Epoch 4/20
118/118 [==============================] - 0s 4ms/step - loss: 0.5945 - accuracy: 0.8667 - val_loss: 0.5311 - val_accuracy: 0.8827
Epoch 5/20
118/118 [==============================] - 0s 4ms/step - loss: 0.5198 - accuracy: 0.8764 - val_loss: 0.4749 - val_accuracy: 0.8885
Epoch 6/20
118/118 [==============================] - 0s 4ms/step - loss: 0.4717 - accuracy: 0.8824 - val_loss: 0.4349 - val_accuracy: 0.8955
Epoch 7/20
118/118 [==============================] - 1s 5ms/step - loss: 0.4380 - accuracy: 0.8869 - val_loss: 0.4078 - val_accuracy: 0.8976
Epoch 8/20
118/118 [==============================] - 0s 4ms/step - loss: 0.4131 - accuracy: 0.8912 - val_loss: 0.3864 - val_accuracy: 0.9005
Epoch 9/20
118/118 [==============================] - 0s 4ms/step - loss: 0.3939 - accuracy: 0.8947 - val_loss: 0.3701 - val_accuracy: 0.9029
Epoch 10/20
118/118 [==============================] - 0s 4ms/step - loss: 0.3786 - accuracy: 0.8970 - val_loss: 0.3562 - val_accuracy: 0.9052
Epoch 11/20
118/118 [==============================] - 0s 4ms/step - loss: 0.3658 - accuracy: 0.9001 - val_loss: 0.3446 - val_accuracy: 0.9066
Epoch 12/20
118/118 [==============================] - 0s 4ms/step - loss: 0.3550 - accuracy: 0.9023 - val_loss: 0.3355 - val_accuracy: 0.9082
Epoch 13/20
118/118 [==============================] - 0s 4ms/step - loss: 0.3458 - accuracy: 0.9038 - val_loss: 0.3273 - val_accuracy: 0.9105
Epoch 14/20
118/118 [==============================] - 0s 4ms/step - loss: 0.3376 - accuracy: 0.9056 - val_loss: 0.3203 - val_accuracy: 0.9115
Epoch 15/20
118/118 [==============================] - 0s 4ms/step - loss: 0.3306 - accuracy: 0.9074 - val_loss: 0.3140 - val_accuracy: 0.9114
Epoch 16/20
118/118 [==============================] - 0s 4ms/step - loss: 0.3241 - accuracy: 0.9088 - val_loss: 0.3087 - val_accuracy: 0.9136
Epoch 17/20
118/118 [==============================] - 0s 4ms/step - loss: 0.3182 - accuracy: 0.9105 - val_loss: 0.3032 - val_accuracy: 0.9147
Epoch 18/20
118/118 [==============================] - 1s 4ms/step - loss: 0.3129 - accuracy: 0.9118 - val_loss: 0.2984 - val_accuracy: 0.9156
Epoch 19/20
118/118 [==============================] - 1s 6ms/step - loss: 0.3078 - accuracy: 0.9129 - val_loss: 0.2940 - val_accuracy: 0.9166
Epoch 20/20
118/118 [==============================] - 1s 6ms/step - loss: 0.3033 - accuracy: 0.9145 - val_loss: 0.2898 - val_accuracy: 0.9179
```

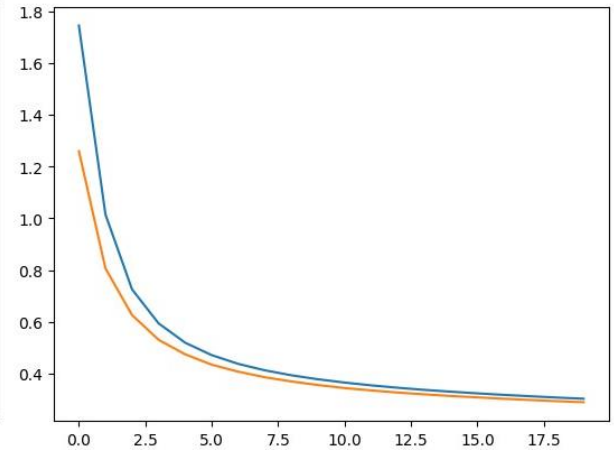
Evaluating model performance

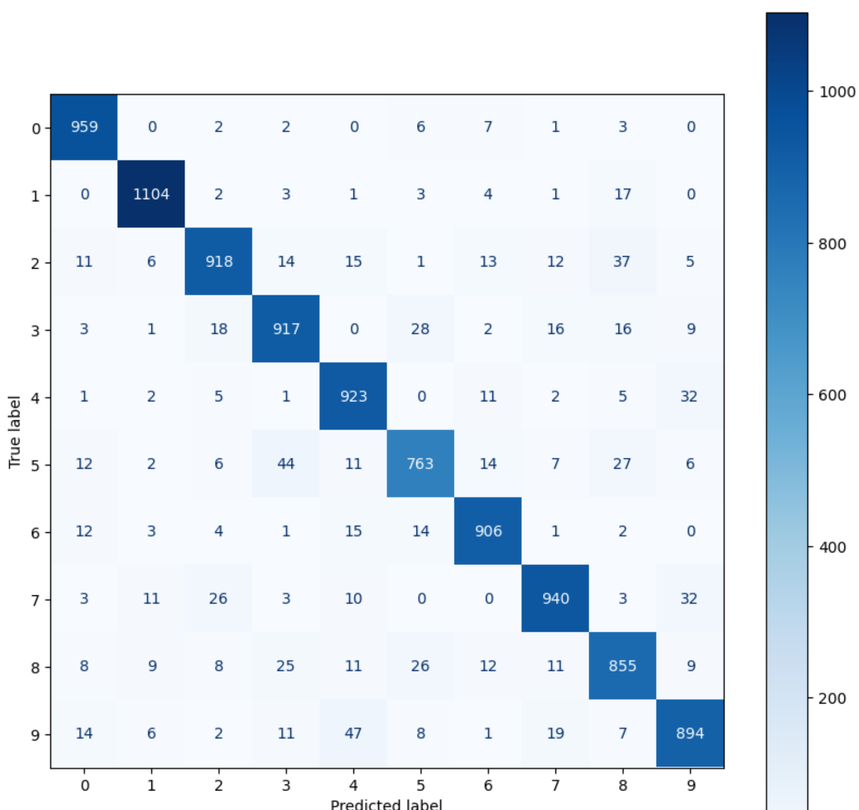
1s

```
print(X_valid.shape)
print(y_valid.shape)
# evaluate the model using the entire validation data set
model.evaluate(X_valid, y_valid)
#from sklearn.metrics import ConfusionMatrixDisplay
#from sklearn.metrics import confusion_matrix
#labels = [str(digit) for digit in range(10)]
#y_pred = np.array(y_predicted)
#full_cm = confusion_matrix(y_valid, y_pred)
#y_pred.shape
```

```
(10000, 784)
(10000, 10)
313/313 [==============================] - 1s 3ms/step - loss: 0.2898 - accuracy: 0.9179
[0.2897818088531494, 0.917900025844574]
```

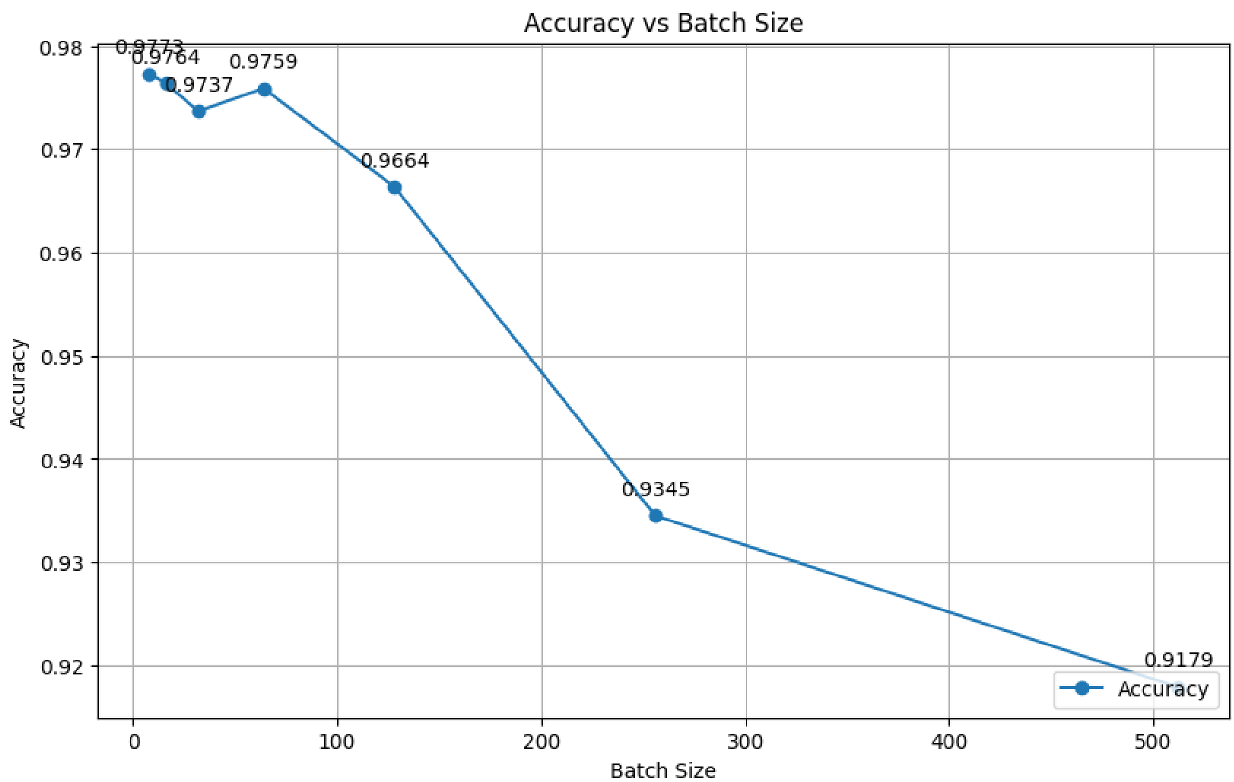





True label
Predicted label

Results Visualization

Batch Size	512	256	128	64	32	16	8
Accuracy	0.9179	0.9345	0.9664	.9759	0.9737	0.9764	0.9773



The chart illustrates a notable increase in accuracy when comparing batch sizes of 500 to those of 64 or even 128. It is evident that, in this model, opting for smaller batch sizes is advisable, as doing so no longer compromises accuracy at a satisfying rate. However, it is prudent to choose a sufficiently small batch size, as beyond a certain threshold, the increase in accuracy becomes barely noticeable. To my believe batch size 64 is that threshold value.

However it is worth noticing that I have ben operating under the assumption that number of epochs is constant. By increasing the number of them the chart would flatten and differences would not be this vivid. Therefore, while considering the batch size, it is necessary to include into estimation all of following factors: Number of epochs

Impact of Batch Size on Training

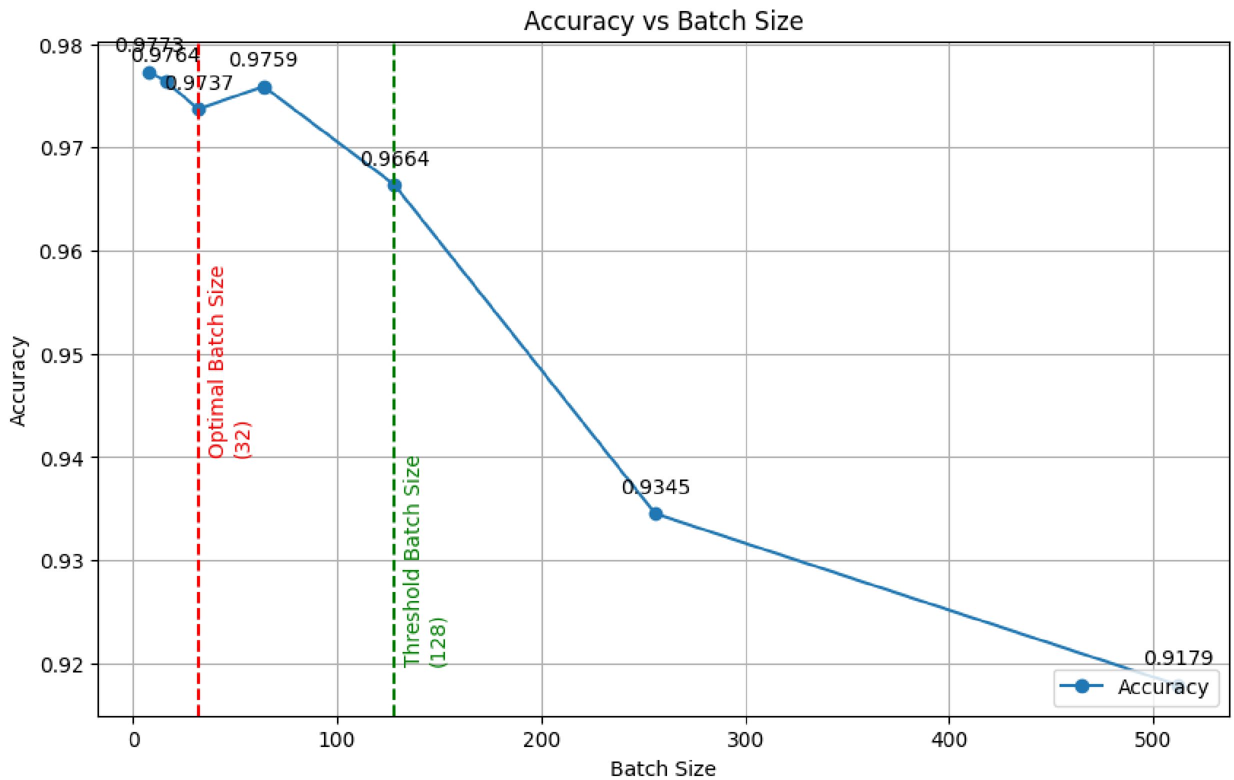
- **Larger Batch Size:**
  - Pros: Faster training due to fewer parameter updates per epoch.
  - Cons: Less accurate gradient estimates due to averaging over more samples, potentially leading to getting stuck in local minima and reducing generalization. Requires more memory on your hardware to store the batch.
- **Smaller Batch Size:**
  - Pros: More accurate gradient estimates due to less averaging, potentially improving generalization. Regularizes the model by adding noise to the training process, helping to prevent overfitting.
  - Cons: Slower training due to more parameter updates per epoch.

Having all gather data, it might be possible to estimate the batch size in the end it is worth sticking with.

```python
batch_sizes = [512, 256, 128, 64, 32, 16, 8]
accuracies = [0.9179, 0.9345, 0.9664, 0.9759, 0.9737, 0.9764, 0.9773]

# Calculate accuracy changes
accuracy_changes = np.diff(accuracies)  # Difference between consecutive elements

# Print accuracy changes with corresponding batch sizes (except the last one)
for i, change in enumerate(accuracy_changes):
  print(f"Change in accuracy from {batch_sizes[i+1]} to {batch_sizes[i]}: {change}")

# Define a threshold for considering accuracy change insignificant
threshold = 0.001  # You can adjust this value based on your needs

# Identify potential threshold point (first change less than or equal to threshold)
potential_threshold_index = np.where(accuracy_changes <= threshold)[0][0] if len(accuracy_changes[accuracy_changes <= threshold]) > 0 else None

if potential_threshold_index is not None:
  potential_threshold_batch_size = batch_sizes[potential_threshold_index + 1]
  print(f"\nPotential threshold for insignificant accuracy change: Batch size {potential_threshold_batch_size} with accuracy {accuracies[potential_threshold_index + 1]}")
else:
  print("\nNo significant accuracy change threshold identified within the provided data.")
```

```
Change in accuracy from 256 to 512: 0.016599999999999948
Change in accuracy from 128 to 256: 0.03190000000000004
Change in accuracy from 64 to 128: 0.009499999999999953
Change in accuracy from 32 to 64: -0.0021999999999999797
Change in accuracy from 16 to 32: 0.0027000000000000357
Change in accuracy from 8 to 16: 0.0008999999999999009

Potential threshold for insignificant accuracy change: Batch size 32 with accuracy 0.9737
```

After conducting thorough analysis, it is apparent that batch sizes ranging from 32 to 128 exhibit favorable performance, with the optimal choice likely influenced by additional contextual factors.



Learning Rate

The learning rate in machine learning refers to a hyperparameter that determines the size of steps taken during the optimization process of model training. It plays a crucial role in balancing the trade-off between the speed of convergence and the risk of overshooting optimal values.

During model training, I experimented with several arbitrarily chosen values such as 0.001, 0.01, 0.5, and 0.1, adjusting towards the one with higher accuracy by a small margin each time.

It is also important to note that while some learning rate values yield high accuracy, the disparity between training and validation accuracy may be too significant for the model to be deemed reliable.

Loss Functions

- binary_crossentropy - Classsification with only 2 classes
- categorical_crossentropy - Classification with multiple classes
- mean_squared_error - Regression

```
# model.compile(loss='mean_squared_error', optimizer=SGD(learning_rate=0.01), metrics=['accuracy'])       # 0.57 accuracy
# model.compile(loss='categorical_crossentropy', optimizer=SGD(learning_rate=0.01), metrics=['accuracy']) # 0.91 accuracy
# model.compile(loss='categorical_crossentropy', optimizer=SGD(learning_rate=0.5), metrics=['accuracy'])  # 0.98 accuracy but diffrence between training accuracy and validationa accuracy is big
model.compile(loss='categorical_crossentropy', optimizer=SGD(learning_rate=0.1), metrics=['accuracy'])    # 0.986 accuracy and validation is close enough
```

Training and result for different Activation function.

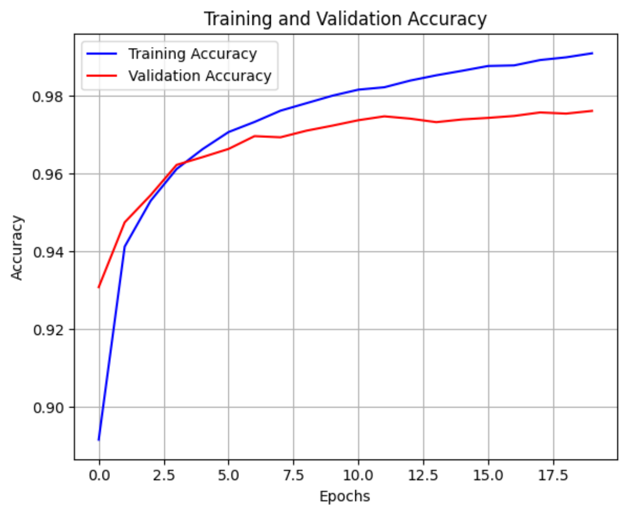
Since fining the right batch sizes I proceeded all following experiments only for them.

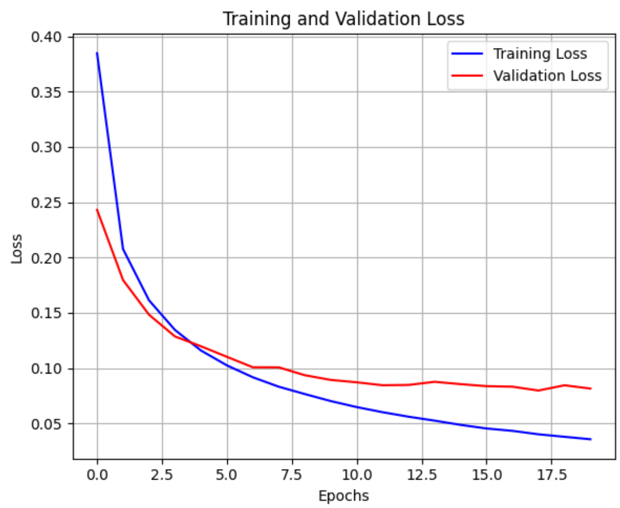
Batch size 64

ReLU (Rectified Linear Unit)

```
evolution_relu = model_relu.fit(X_train, y_train, batch_size=64, epochs=20, verbose=1, validation_data=(X_valid, y_valid))
evolution_tanh = model_tanh.fit(X_train, y_train, batch_size=64, epochs=20, verbose=1, validation_data=(X_valid, y_valid))
```

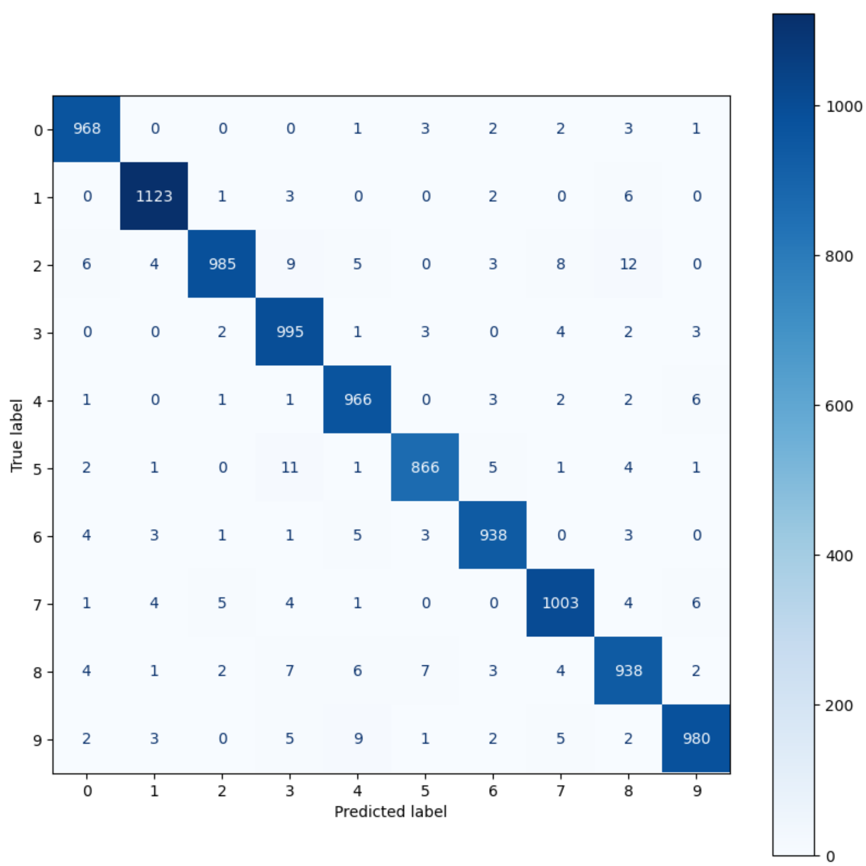
```
Epoch 1/20
938/938 [==============================] - 3s 3ms/step - loss: 0.3847 - accuracy: 0.8915 - val_loss: 0.2431 - val_accuracy: 0.9308
Epoch 2/20
938/938 [==============================] - 3s 4ms/step - loss: 0.2077 - accuracy: 0.9413 - val_loss: 0.1797 - val_accuracy: 0.9475
Epoch 3/20
938/938 [==============================] - 2s 3ms/step - loss: 0.1614 - accuracy: 0.9531 - val_loss: 0.1484 - val_accuracy: 0.9545
Epoch 4/20
938/938 [==============================] - 2s 3ms/step - loss: 0.1346 - accuracy: 0.9612 - val_loss: 0.1287 - val_accuracy: 0.9623
Epoch 5/20
938/938 [==============================] - 2s 2ms/step - loss: 0.1160 - accuracy: 0.9664 - val_loss: 0.1198 - val_accuracy: 0.9643
Epoch 6/20
938/938 [==============================] - 3s 3ms/step - loss: 0.1025 - accuracy: 0.9707 - val_loss: 0.1103 - val_accuracy: 0.9664
Epoch 7/20
938/938 [==============================] - 3s 3ms/step - loss: 0.0918 - accuracy: 0.9733 - val_loss: 0.1008 - val_accuracy: 0.9697
Epoch 8/20
938/938 [==============================] - 2s 2ms/step - loss: 0.0832 - accuracy: 0.9762 - val_loss: 0.1007 - val_accuracy: 0.9694
Epoch 9/20
938/938 [==============================] - 2s 3ms/step - loss: 0.0766 - accuracy: 0.9782 - val_loss: 0.0936 - val_accuracy: 0.9711
Epoch 10/20
938/938 [==============================] - 2s 3ms/step - loss: 0.0703 - accuracy: 0.9801 - val_loss: 0.0894 - val_accuracy: 0.9724
Epoch 11/20
938/938 [==============================] - 3s 3ms/step - loss: 0.0648 - accuracy: 0.9817 - val_loss: 0.0872 - val_accuracy: 0.9738
Epoch 12/20
938/938 [==============================] - 3s 3ms/step - loss: 0.0602 - accuracy: 0.9823 - val_loss: 0.0846 - val_accuracy: 0.9748
Epoch 13/20
938/938 [==============================] - 2s 3ms/step - loss: 0.0561 - accuracy: 0.9840 - val_loss: 0.0849 - val_accuracy: 0.9742
Epoch 14/20
938/938 [==============================] - 2s 3ms/step - loss: 0.0526 - accuracy: 0.9854 - val_loss: 0.0877 - val_accuracy: 0.9733
Epoch 15/20
938/938 [==============================] - 2s 3ms/step - loss: 0.0487 - accuracy: 0.9865 - val_loss: 0.0856 - val_accuracy: 0.9740
Epoch 16/20
938/938 [==============================] - 3s 4ms/step - loss: 0.0455 - accuracy: 0.9878 - val_loss: 0.0837 - val_accuracy: 0.9744
Epoch 17/20
938/938 [==============================] - 2s 3ms/step - loss: 0.0433 - accuracy: 0.9879 - val_loss: 0.0833 - val_accuracy: 0.9749
Epoch 18/20
938/938 [==============================] - 2s 3ms/step - loss: 0.0402 - accuracy: 0.9893 - val_loss: 0.0798 - val_accuracy: 0.9758
Epoch 19/20
938/938 [==============================] - 2s 2ms/step - loss: 0.0380 - accuracy: 0.9900 - val_loss: 0.0845 - val_accuracy: 0.9755
Epoch 20/20
938/938 [==============================] - 2s 2ms/step - loss: 0.0357 - accuracy: 0.9910 - val_loss: 0.0816 - val_accuracy: 0.9762
```





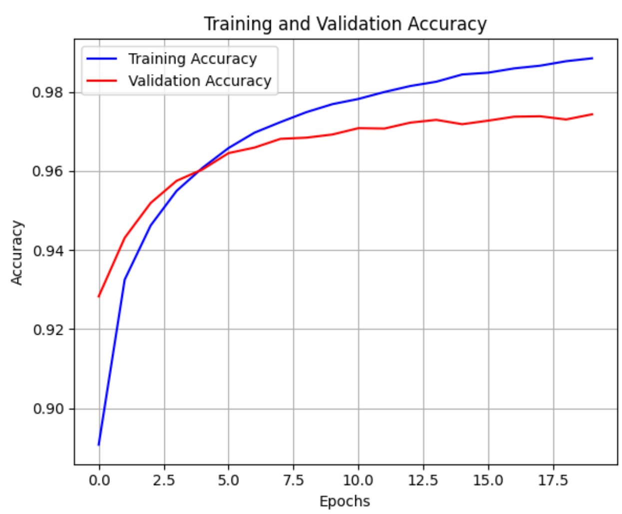

```
# evaluate the model using the entire validation data set
model_relu.evaluate(X_valid, y_valid)
```

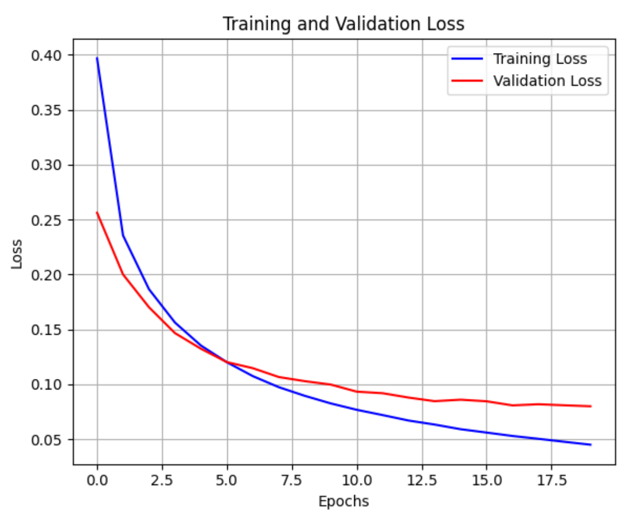
```
313/313 [==============================] - 1s 2ms/step - loss: 0.0816 - accuracy: 0.9762
[0.08160426467657089, 0.9761999845504761]
```



Tanh – Hyperbolic Tangent

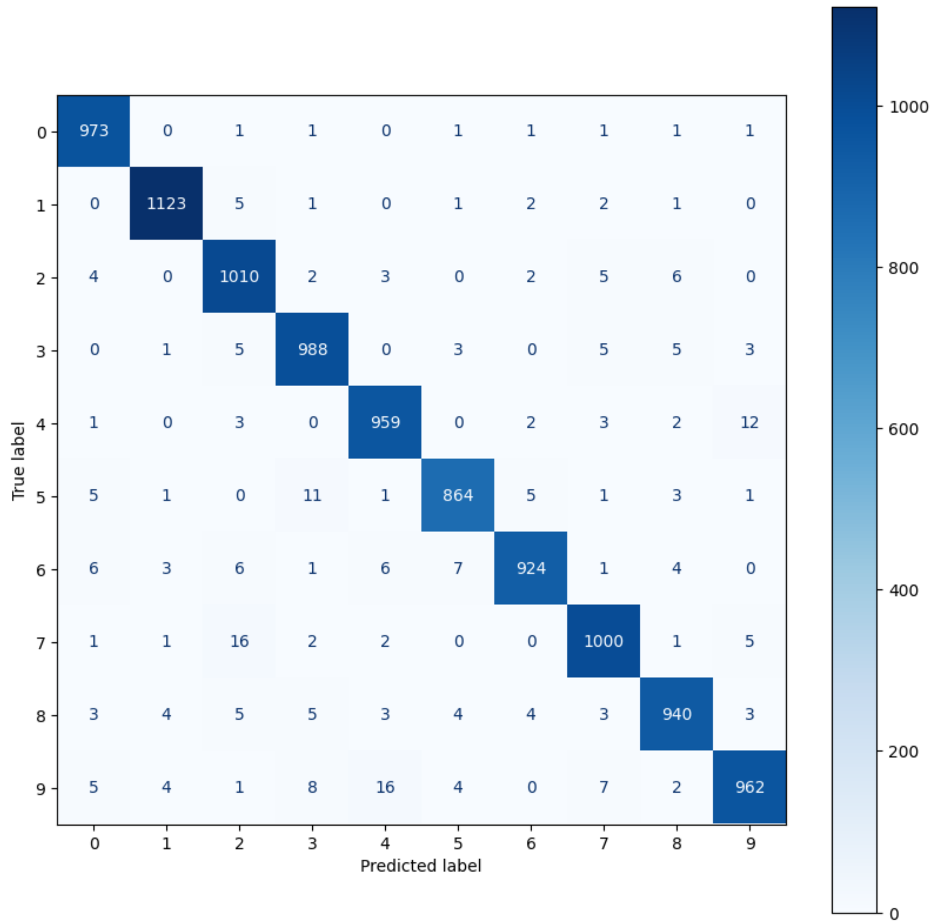
```
Epoch 1/20
938/938 [==============================] - 3s 3ms/step - loss: 0.3968 - accuracy: 0.8908 - val_loss: 0.2562 - val_accuracy: 0.9283
Epoch 2/20
938/938 [==============================] - 3s 3ms/step - loss: 0.2355 - accuracy: 0.9325 - val_loss: 0.2000 - val_accuracy: 0.9431
Epoch 3/20
938/938 [==============================] - 4s 4ms/step - loss: 0.1865 - accuracy: 0.9462 - val_loss: 0.1701 - val_accuracy: 0.9519
Epoch 4/20
938/938 [==============================] - 2s 3ms/step - loss: 0.1562 - accuracy: 0.9550 - val_loss: 0.1466 - val_accuracy: 0.9575
Epoch 5/20
938/938 [==============================] - 2s 3ms/step - loss: 0.1353 - accuracy: 0.9608 - val_loss: 0.1324 - val_accuracy: 0.9604
Epoch 6/20
938/938 [==============================] - 2s 3ms/step - loss: 0.1201 - accuracy: 0.9658 - val_loss: 0.1202 - val_accuracy: 0.9645
Epoch 7/20
938/938 [==============================] - 3s 4ms/step - loss: 0.1074 - accuracy: 0.9697 - val_loss: 0.1146 - val_accuracy: 0.9659
Epoch 8/20
938/938 [==============================] - 2s 3ms/step - loss: 0.0974 - accuracy: 0.9723 - val_loss: 0.1066 - val_accuracy: 0.9681
Epoch 9/20
938/938 [==============================] - 2s 3ms/step - loss: 0.0895 - accuracy: 0.9748 - val_loss: 0.1028 - val_accuracy: 0.9684
Epoch 10/20
938/938 [==============================] - 2s 3ms/step - loss: 0.0826 - accuracy: 0.9769 - val_loss: 0.0997 - val_accuracy: 0.9692
Epoch 11/20
938/938 [==============================] - 2s 3ms/step - loss: 0.0768 - accuracy: 0.9782 - val_loss: 0.0933 - val_accuracy: 0.9708
Epoch 12/20
938/938 [==============================] - 3s 4ms/step - loss: 0.0719 - accuracy: 0.9799 - val_loss: 0.0919 - val_accuracy: 0.9707
Epoch 13/20
938/938 [==============================] - 2s 3ms/step - loss: 0.0670 - accuracy: 0.9815 - val_loss: 0.0879 - val_accuracy: 0.9722
Epoch 14/20
938/938 [==============================] - 2s 3ms/step - loss: 0.0633 - accuracy: 0.9825 - val_loss: 0.0846 - val_accuracy: 0.9729
Epoch 15/20
938/938 [==============================] - 3s 3ms/step - loss: 0.0591 - accuracy: 0.9844 - val_loss: 0.0860 - val_accuracy: 0.9718
Epoch 16/20
938/938 [==============================] - 3s 3ms/step - loss: 0.0560 - accuracy: 0.9848 - val_loss: 0.0845 - val_accuracy: 0.9727
Epoch 17/20
938/938 [==============================] - 3s 3ms/step - loss: 0.0529 - accuracy: 0.9859 - val_loss: 0.0808 - val_accuracy: 0.9737
Epoch 18/20
938/938 [==============================] - 2s 3ms/step - loss: 0.0503 - accuracy: 0.9866 - val_loss: 0.0818 - val_accuracy: 0.9738
Epoch 19/20
938/938 [==============================] - 3s 3ms/step - loss: 0.0476 - accuracy: 0.9877 - val_loss: 0.0809 - val_accuracy: 0.9730
Epoch 20/20
938/938 [==============================] - 2s 3ms/step - loss: 0.0450 - accuracy: 0.9884 - val_loss: 0.0800 - val_accuracy: 0.9743
```





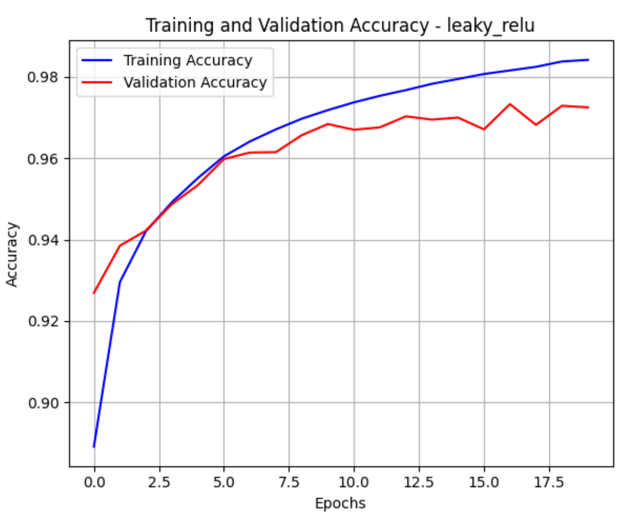
```
# evaluate the model using the entire validation data set
model_tanh.evaluate(X_valid, y_valid)
```

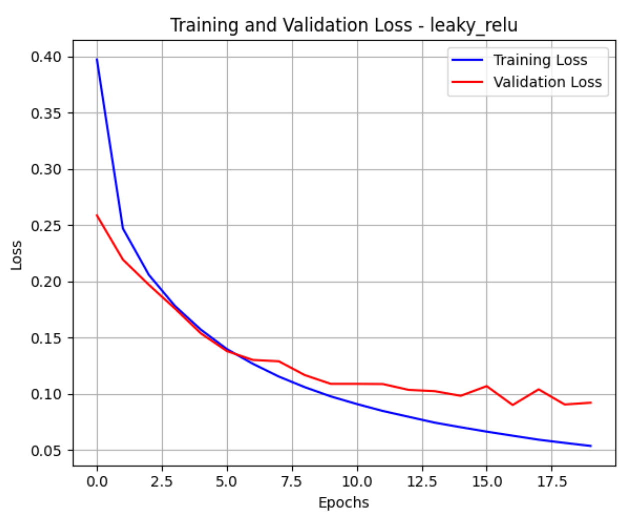
```
313/313 [==============================] - 1s 2ms/step - loss: 0.0800 - accuracy: 0.9743
[0.07998695224523544, 0.9743000268936157]
```



Leaky ReLU

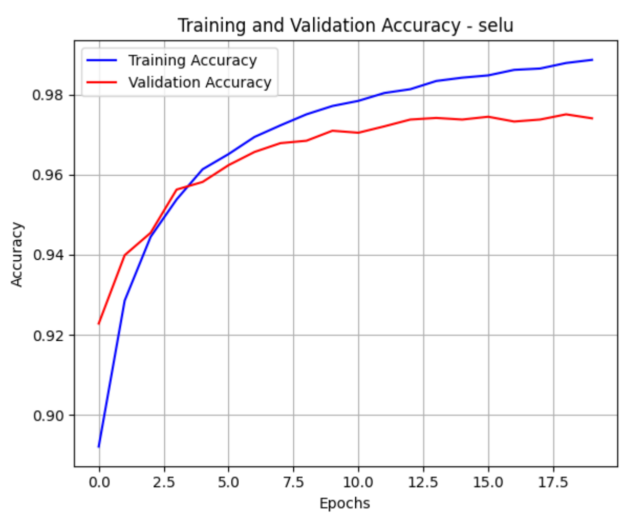
```
••• Training model with activation function: leaky_relu
    Epoch 1/20
    938/938 [==============================] - 8s 8ms/step - loss: 0.3972 - accuracy: 0.8891 - val_loss: 0.2586 - val_accuracy: 0.9269
    Epoch 2/20
    938/938 [==============================] - 3s 4ms/step - loss: 0.2470 - accuracy: 0.9296 - val_loss: 0.2193 - val_accuracy: 0.9385
    Epoch 3/20
    938/938 [==============================] - 2s 3ms/step - loss: 0.2057 - accuracy: 0.9420 - val_loss: 0.1969 - val_accuracy: 0.9422
    Epoch 4/20
    938/938 [==============================] - 3s 4ms/step - loss: 0.1781 - accuracy: 0.9492 - val_loss: 0.1758 - val_accuracy: 0.9487
    Epoch 5/20
    938/938 [==============================] - 4s 5ms/step - loss: 0.1570 - accuracy: 0.9551 - val_loss: 0.1537 - val_accuracy: 0.9534
    Epoch 6/20
    938/938 [==============================] - 3s 3ms/step - loss: 0.1397 - accuracy: 0.9605 - val_loss: 0.1379 - val_accuracy: 0.9598
    Epoch 7/20
    938/938 [==============================] - 3s 3ms/step - loss: 0.1266 - accuracy: 0.9641 - val_loss: 0.1301 - val_accuracy: 0.9614
    Epoch 8/20
    938/938 [==============================] - 4s 4ms/step - loss: 0.1153 - accuracy: 0.9671 - val_loss: 0.1288 - val_accuracy: 0.9615
    Epoch 9/20
    938/938 [==============================] - 3s 3ms/step - loss: 0.1058 - accuracy: 0.9697 - val_loss: 0.1166 - val_accuracy: 0.9657
    Epoch 10/20
    938/938 [==============================] - 3s 4ms/step - loss: 0.0976 - accuracy: 0.9718 - val_loss: 0.1088 - val_accuracy: 0.9684
    Epoch 11/20
    938/938 [==============================] - 4s 4ms/step - loss: 0.0908 - accuracy: 0.9737 - val_loss: 0.1088 - val_accuracy: 0.9670
    Epoch 12/20
    938/938 [==============================] - 4s 4ms/step - loss: 0.0847 - accuracy: 0.9754 - val_loss: 0.1086 - val_accuracy: 0.9676
    Epoch 13/20
    938/938 [==============================] - 3s 3ms/step - loss: 0.0795 - accuracy: 0.9768 - val_loss: 0.1034 - val_accuracy: 0.9703
    Epoch 14/20
    938/938 [==============================] - 3s 4ms/step - loss: 0.0743 - accuracy: 0.9783 - val_loss: 0.1023 - val_accuracy: 0.9695
    Epoch 15/20
    938/938 [==============================] - 4s 5ms/step - loss: 0.0702 - accuracy: 0.9795 - val_loss: 0.0981 - val_accuracy: 0.9700
    Epoch 16/20
    938/938 [==============================] - 3s 3ms/step - loss: 0.0663 - accuracy: 0.9807 - val_loss: 0.1067 - val_accuracy: 0.9671
    Epoch 17/20
    938/938 [==============================] - 3s 3ms/step - loss: 0.0627 - accuracy: 0.9816 - val_loss: 0.0900 - val_accuracy: 0.9733
    Epoch 18/20
    938/938 [==============================] - 3s 3ms/step - loss: 0.0591 - accuracy: 0.9825 - val_loss: 0.1039 - val_accuracy: 0.9682
    Epoch 19/20
    938/938 [==============================] - 3s 4ms/step - loss: 0.0563 - accuracy: 0.9838 - val_loss: 0.0904 - val_accuracy: 0.9729
    Epoch 20/20
    938/938 [==============================] - 4s 4ms/step - loss: 0.0535 - accuracy: 0.9842 - val_loss: 0.0920 - val_accuracy: 0.9725
    Training complete for model with activation function: leaky_relu
```

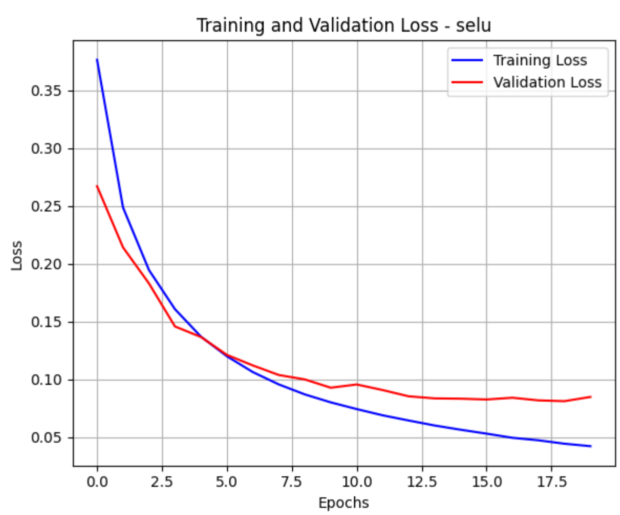




SeLU

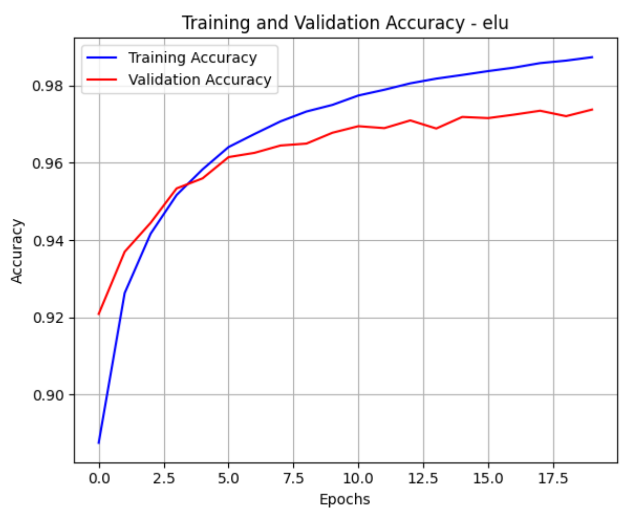
```
938/938 [==============================] - 4s 4ms/step - loss: 0.3762 - accuracy: 0.8921 - val_loss: 0.2669 - val_accuracy: 0.9228
Epoch 2/20
938/938 [==============================] - 4s 4ms/step - loss: 0.2484 - accuracy: 0.9286 - val_loss: 0.2140 - val_accuracy: 0.9399
Epoch 3/20
938/938 [==============================] - 4s 4ms/step - loss: 0.1944 - accuracy: 0.9445 - val_loss: 0.1828 - val_accuracy: 0.9455
Epoch 4/20
938/938 [==============================] - 3s 4ms/step - loss: 0.1606 - accuracy: 0.9538 - val_loss: 0.1456 - val_accuracy: 0.9563
Epoch 5/20
938/938 [==============================] - 3s 3ms/step - loss: 0.1368 - accuracy: 0.9614 - val_loss: 0.1365 - val_accuracy: 0.9582
Epoch 6/20
938/938 [==============================] - 4s 4ms/step - loss: 0.1199 - accuracy: 0.9652 - val_loss: 0.1210 - val_accuracy: 0.9624
Epoch 7/20
938/938 [==============================] - 4s 4ms/step - loss: 0.1062 - accuracy: 0.9694 - val_loss: 0.1119 - val_accuracy: 0.9657
Epoch 8/20
938/938 [==============================] - 3s 3ms/step - loss: 0.0955 - accuracy: 0.9724 - val_loss: 0.1036 - val_accuracy: 0.9679
Epoch 9/20
938/938 [==============================] - 3s 3ms/step - loss: 0.0869 - accuracy: 0.9751 - val_loss: 0.0998 - val_accuracy: 0.9685
Epoch 10/20
938/938 [==============================] - 4s 4ms/step - loss: 0.0800 - accuracy: 0.9772 - val_loss: 0.0926 - val_accuracy: 0.9710
Epoch 11/20
938/938 [==============================] - 3s 3ms/step - loss: 0.0742 - accuracy: 0.9785 - val_loss: 0.0954 - val_accuracy: 0.9705
Epoch 12/20
938/938 [==============================] - 3s 3ms/step - loss: 0.0688 - accuracy: 0.9804 - val_loss: 0.0906 - val_accuracy: 0.9721
Epoch 13/20
938/938 [==============================] - 2s 3ms/step - loss: 0.0643 - accuracy: 0.9814 - val_loss: 0.0852 - val_accuracy: 0.9738
Epoch 14/20
938/938 [==============================] - 3s 3ms/step - loss: 0.0600 - accuracy: 0.9834 - val_loss: 0.0834 - val_accuracy: 0.9742
Epoch 15/20
938/938 [==============================] - 4s 4ms/step - loss: 0.0563 - accuracy: 0.9843 - val_loss: 0.0832 - val_accuracy: 0.9738
Epoch 16/20
938/938 [==============================] - 3s 4ms/step - loss: 0.0529 - accuracy: 0.9848 - val_loss: 0.0825 - val_accuracy: 0.9745
Epoch 17/20
938/938 [==============================] - 3s 3ms/step - loss: 0.0493 - accuracy: 0.9862 - val_loss: 0.0840 - val_accuracy: 0.9733
Epoch 18/20
938/938 [==============================] - 4s 4ms/step - loss: 0.0472 - accuracy: 0.9865 - val_loss: 0.0817 - val_accuracy: 0.9738
Epoch 19/20
938/938 [==============================] - 3s 3ms/step - loss: 0.0442 - accuracy: 0.9879 - val_loss: 0.0810 - val_accuracy: 0.9751
Epoch 20/20
938/938 [==============================] - 3s 3ms/step - loss: 0.0421 - accuracy: 0.9887 - val_loss: 0.0847 - val_accuracy: 0.9741
Training complete for model with activation function: selu
```

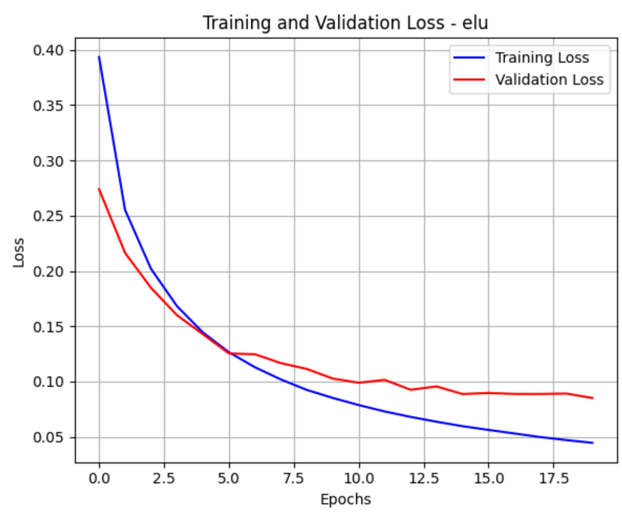




eLU

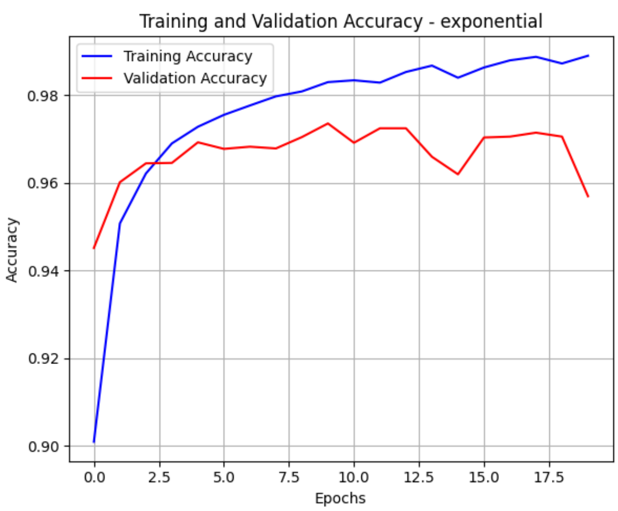
```
Training model with activation function: elu
Epoch 1/20
938/938 [==============================] - 3s 3ms/step - loss: 0.3934 - accuracy: 0.8875 - val_loss: 0.2740 - val_accuracy: 0.9209
Epoch 2/20
938/938 [==============================] - 4s 4ms/step - loss: 0.2553 - accuracy: 0.9263 - val_loss: 0.2165 - val_accuracy: 0.9370
Epoch 3/20
938/938 [==============================] - 3s 3ms/step - loss: 0.2020 - accuracy: 0.9417 - val_loss: 0.1848 - val_accuracy: 0.9445
Epoch 4/20
938/938 [==============================] - 3s 3ms/step - loss: 0.1682 - accuracy: 0.9517 - val_loss: 0.1600 - val_accuracy: 0.9534
Epoch 5/20
938/938 [==============================] - 2s 3ms/step - loss: 0.1444 - accuracy: 0.9583 - val_loss: 0.1428 - val_accuracy: 0.9560
Epoch 6/20
938/938 [==============================] - 4s 4ms/step - loss: 0.1265 - accuracy: 0.9641 - val_loss: 0.1254 - val_accuracy: 0.9615
Epoch 7/20
938/938 [==============================] - 3s 3ms/step - loss: 0.1130 - accuracy: 0.9675 - val_loss: 0.1247 - val_accuracy: 0.9626
Epoch 8/20
938/938 [==============================] - 4s 4ms/step - loss: 0.1019 - accuracy: 0.9707 - val_loss: 0.1167 - val_accuracy: 0.9645
Epoch 9/20
938/938 [==============================] - 4s 4ms/step - loss: 0.0924 - accuracy: 0.9733 - val_loss: 0.1114 - val_accuracy: 0.9650
Epoch 10/20
938/938 [==============================] - 4s 5ms/step - loss: 0.0852 - accuracy: 0.9750 - val_loss: 0.1027 - val_accuracy: 0.9678
Epoch 11/20
938/938 [==============================] - 4s 4ms/step - loss: 0.0787 - accuracy: 0.9775 - val_loss: 0.0989 - val_accuracy: 0.9695
Epoch 12/20
938/938 [==============================] - 3s 4ms/step - loss: 0.0729 - accuracy: 0.9789 - val_loss: 0.1015 - val_accuracy: 0.9690
Epoch 13/20
938/938 [==============================] - 3s 3ms/step - loss: 0.0680 - accuracy: 0.9806 - val_loss: 0.0926 - val_accuracy: 0.9710
Epoch 14/20
938/938 [==============================] - 4s 4ms/step - loss: 0.0636 - accuracy: 0.9818 - val_loss: 0.0956 - val_accuracy: 0.9689
Epoch 15/20
938/938 [==============================] - 3s 3ms/step - loss: 0.0596 - accuracy: 0.9828 - val_loss: 0.0887 - val_accuracy: 0.9719
Epoch 16/20
938/938 [==============================] - 2s 3ms/step - loss: 0.0562 - accuracy: 0.9838 - val_loss: 0.0896 - val_accuracy: 0.9716
Epoch 17/20
938/938 [==============================] - 4s 4ms/step - loss: 0.0530 - accuracy: 0.9847 - val_loss: 0.0888 - val_accuracy: 0.9725
Epoch 18/20
938/938 [==============================] - 4s 4ms/step - loss: 0.0497 - accuracy: 0.9858 - val_loss: 0.0887 - val_accuracy: 0.9735
Epoch 19/20
938/938 [==============================] - 3s 4ms/step - loss: 0.0470 - accuracy: 0.9865 - val_loss: 0.0891 - val_accuracy: 0.9721
Epoch 20/20
938/938 [==============================] - 3s 3ms/step - loss: 0.0445 - accuracy: 0.9874 - val_loss: 0.0851 - val_accuracy: 0.9738
Training complete for model with activation function: elu
```

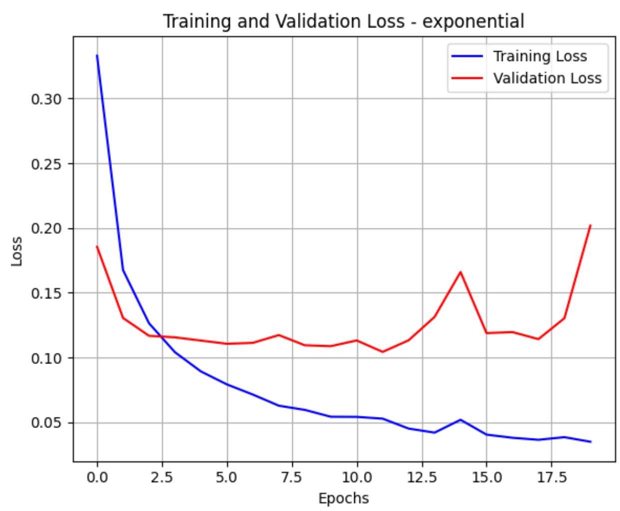




Exponential

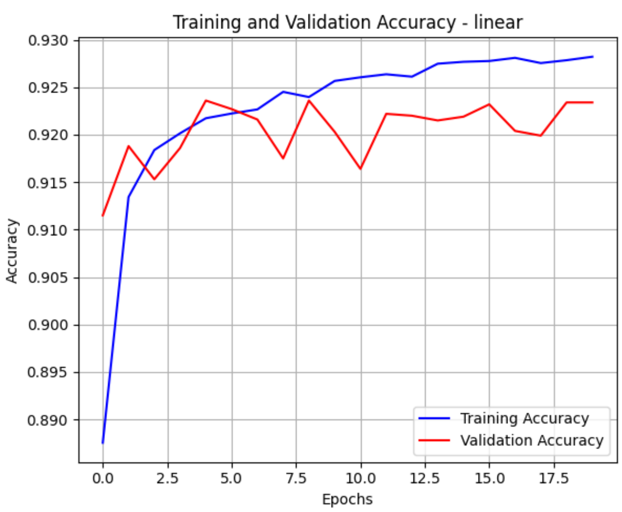
```
Training model with activation function: exponential
Epoch 1/20
938/938 [==============================] - 4s 3ms/step - loss: 0.3329 - accuracy: 0.9010 - val_loss: 0.1855 - val_accuracy: 0.9452
Epoch 2/20
938/938 [==============================] - 3s 3ms/step - loss: 0.1675 - accuracy: 0.9508 - val_loss: 0.1306 - val_accuracy: 0.9602
Epoch 3/20
938/938 [==============================] - 3s 3ms/step - loss: 0.1265 - accuracy: 0.9622 - val_loss: 0.1168 - val_accuracy: 0.9645
Epoch 4/20
938/938 [==============================] - 4s 4ms/step - loss: 0.1041 - accuracy: 0.9690 - val_loss: 0.1156 - val_accuracy: 0.9646
Epoch 5/20
938/938 [==============================] - 3s 3ms/step - loss: 0.0893 - accuracy: 0.9729 - val_loss: 0.1131 - val_accuracy: 0.9693
Epoch 6/20
938/938 [==============================] - 4s 5ms/step - loss: 0.0793 - accuracy: 0.9756 - val_loss: 0.1107 - val_accuracy: 0.9678
Epoch 7/20
938/938 [==============================] - 4s 4ms/step - loss: 0.0715 - accuracy: 0.9777 - val_loss: 0.1114 - val_accuracy: 0.9683
Epoch 8/20
938/938 [==============================] - 3s 3ms/step - loss: 0.0629 - accuracy: 0.9798 - val_loss: 0.1174 - val_accuracy: 0.9679
Epoch 9/20
938/938 [==============================] - 4s 4ms/step - loss: 0.0597 - accuracy: 0.9809 - val_loss: 0.1095 - val_accuracy: 0.9705
Epoch 10/20
938/938 [==============================] - 5s 5ms/step - loss: 0.0544 - accuracy: 0.9830 - val_loss: 0.1088 - val_accuracy: 0.9736
Epoch 11/20
938/938 [==============================] - 3s 3ms/step - loss: 0.0543 - accuracy: 0.9834 - val_loss: 0.1132 - val_accuracy: 0.9692
Epoch 12/20
938/938 [==============================] - 3s 3ms/step - loss: 0.0529 - accuracy: 0.9829 - val_loss: 0.1044 - val_accuracy: 0.9725
Epoch 13/20
938/938 [==============================] - 5s 5ms/step - loss: 0.0452 - accuracy: 0.9854 - val_loss: 0.1134 - val_accuracy: 0.9725
Epoch 14/20
938/938 [==============================] - 3s 3ms/step - loss: 0.0421 - accuracy: 0.9868 - val_loss: 0.1314 - val_accuracy: 0.9660
Epoch 15/20
938/938 [==============================] - 3s 3ms/step - loss: 0.0520 - accuracy: 0.9840 - val_loss: 0.1660 - val_accuracy: 0.9620
Epoch 16/20
938/938 [==============================] - 4s 5ms/step - loss: 0.0406 - accuracy: 0.9864 - val_loss: 0.1189 - val_accuracy: 0.9704
Epoch 17/20
938/938 [==============================] - 3s 3ms/step - loss: 0.0381 - accuracy: 0.9880 - val_loss: 0.1197 - val_accuracy: 0.9706
Epoch 18/20
938/938 [==============================] - 4s 4ms/step - loss: 0.0366 - accuracy: 0.9888 - val_loss: 0.1142 - val_accuracy: 0.9715
Epoch 19/20
938/938 [==============================] - 3s 3ms/step - loss: 0.0386 - accuracy: 0.9873 - val_loss: 0.1304 - val_accuracy: 0.9706
Epoch 20/20
938/938 [==============================] - 4s 4ms/step - loss: 0.0351 - accuracy: 0.9890 - val_loss: 0.2018 - val_accuracy: 0.9570
Training complete for model with activation function: exponential
```

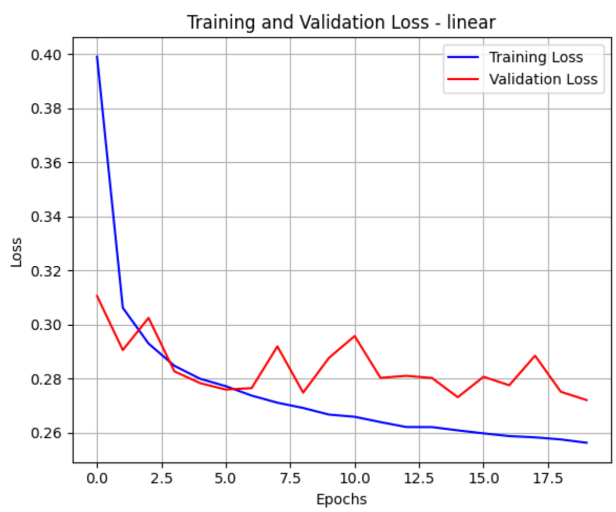




Linear

```
Training model with activation function: linear
Epoch 1/20
938/938 [==============================] - 5s 4ms/step - loss: 0.3991 - accuracy: 0.8875 - val_loss: 0.3106 - val_accuracy: 0.9115
Epoch 2/20
938/938 [==============================] - 6s 6ms/step - loss: 0.3062 - accuracy: 0.9134 - val_loss: 0.2906 - val_accuracy: 0.9188
Epoch 3/20
938/938 [==============================] - 4s 4ms/step - loss: 0.2930 - accuracy: 0.9184 - val_loss: 0.3025 - val_accuracy: 0.9153
Epoch 4/20
938/938 [==============================] - 4s 4ms/step - loss: 0.2848 - accuracy: 0.9201 - val_loss: 0.2827 - val_accuracy: 0.9186
Epoch 5/20
938/938 [==============================] - 3s 3ms/step - loss: 0.2800 - accuracy: 0.9217 - val_loss: 0.2784 - val_accuracy: 0.9236
Epoch 6/20
938/938 [==============================] - 4s 4ms/step - loss: 0.2772 - accuracy: 0.9222 - val_loss: 0.2760 - val_accuracy: 0.9227
Epoch 7/20
938/938 [==============================] - 3s 4ms/step - loss: 0.2738 - accuracy: 0.9227 - val_loss: 0.2765 - val_accuracy: 0.9216
Epoch 8/20
938/938 [==============================] - 3s 3ms/step - loss: 0.2711 - accuracy: 0.9245 - val_loss: 0.2920 - val_accuracy: 0.9175
Epoch 9/20
938/938 [==============================] - 3s 3ms/step - loss: 0.2692 - accuracy: 0.9240 - val_loss: 0.2748 - val_accuracy: 0.9236
Epoch 10/20
938/938 [==============================] - 4s 4ms/step - loss: 0.2667 - accuracy: 0.9257 - val_loss: 0.2877 - val_accuracy: 0.9203
Epoch 11/20
938/938 [==============================] - 4s 4ms/step - loss: 0.2659 - accuracy: 0.9261 - val_loss: 0.2958 - val_accuracy: 0.9164
Epoch 12/20
938/938 [==============================] - 3s 3ms/step - loss: 0.2640 - accuracy: 0.9264 - val_loss: 0.2803 - val_accuracy: 0.9222
Epoch 13/20
938/938 [==============================] - 3s 3ms/step - loss: 0.2621 - accuracy: 0.9261 - val_loss: 0.2811 - val_accuracy: 0.9220
Epoch 14/20
938/938 [==============================] - 4s 4ms/step - loss: 0.2621 - accuracy: 0.9275 - val_loss: 0.2803 - val_accuracy: 0.9215
Epoch 15/20
938/938 [==============================] - 4s 5ms/step - loss: 0.2609 - accuracy: 0.9277 - val_loss: 0.2731 - val_accuracy: 0.9219
Epoch 16/20
938/938 [==============================] - 3s 3ms/step - loss: 0.2598 - accuracy: 0.9278 - val_loss: 0.2807 - val_accuracy: 0.9232
Epoch 17/20
938/938 [==============================] - 4s 4ms/step - loss: 0.2588 - accuracy: 0.9281 - val_loss: 0.2776 - val_accuracy: 0.9204
Epoch 18/20
938/938 [==============================] - 3s 3ms/step - loss: 0.2583 - accuracy: 0.9276 - val_loss: 0.2885 - val_accuracy: 0.9199
Epoch 19/20
938/938 [==============================] - 3s 3ms/step - loss: 0.2575 - accuracy: 0.9279 - val_loss: 0.2752 - val_accuracy: 0.9234
Epoch 20/20
938/938 [==============================] - 3s 3ms/step - loss: 0.2563 - accuracy: 0.9282 - val_loss: 0.2721 - val_accuracy: 0.9234
```



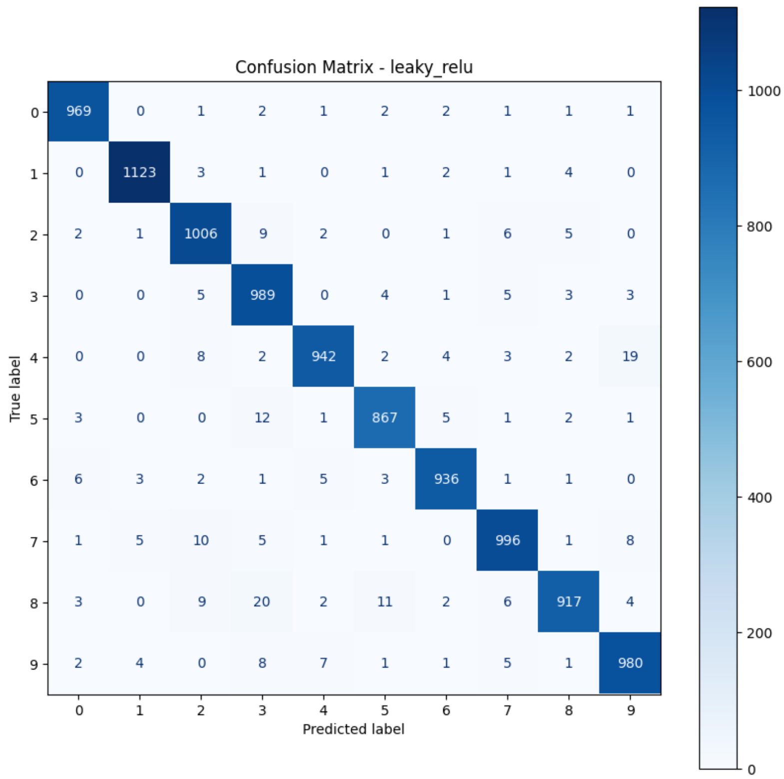


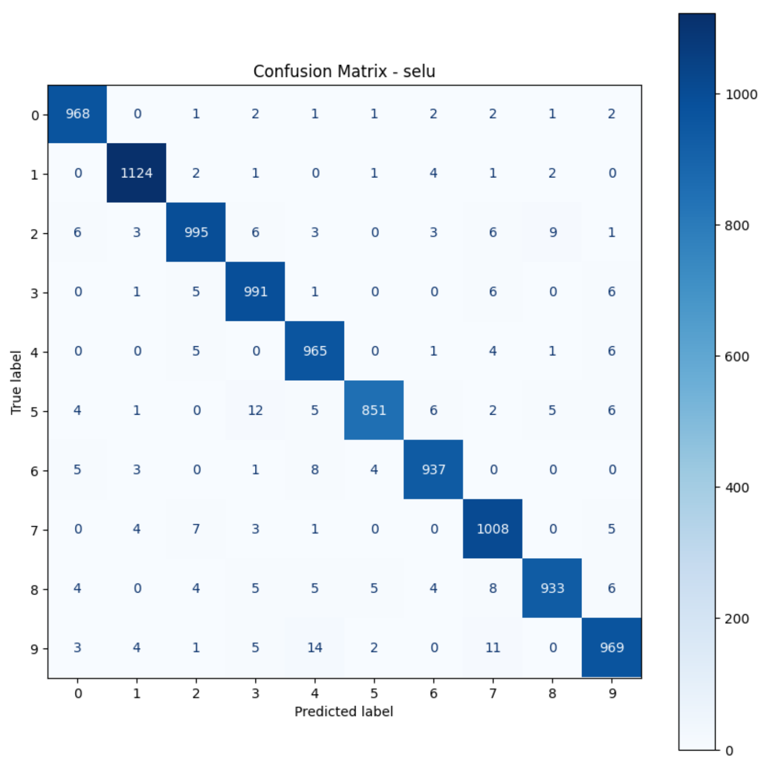
Evaluation

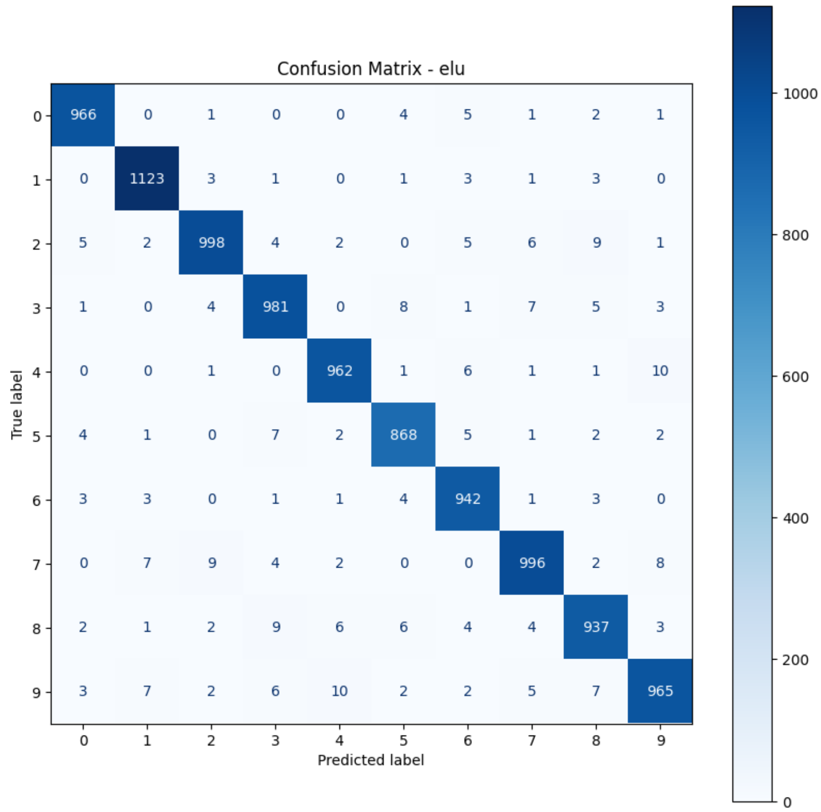
```
Evaluating model with activation function: leaky_relu
313/313 [==============================] - 2s 8ms/step - loss: 0.0920 - accuracy: 0.9725
Evaluating complete for model with activation function: leaky_relu


Evaluating model with activation function: selu
313/313 [==============================] - 2s 5ms/step - loss: 0.0847 - accuracy: 0.9741
Evaluating complete for model with activation function: selu


Evaluating model with activation function: elu
313/313 [==============================] - 1s 2ms/step - loss: 0.0851 - accuracy: 0.9738
Evaluating complete for model with activation function: elu


Evaluating model with activation function: exponential
313/313 [==============================] - 1s 2ms/step - loss: 0.2018 - accuracy: 0.9570
Evaluating complete for model with activation function: exponential


Evaluating model with activation function: linear
313/313 [==============================] - 1s 2ms/step - loss: 0.2721 - accuracy: 0.9234
Evaluating complete for model with activation function: linear
```

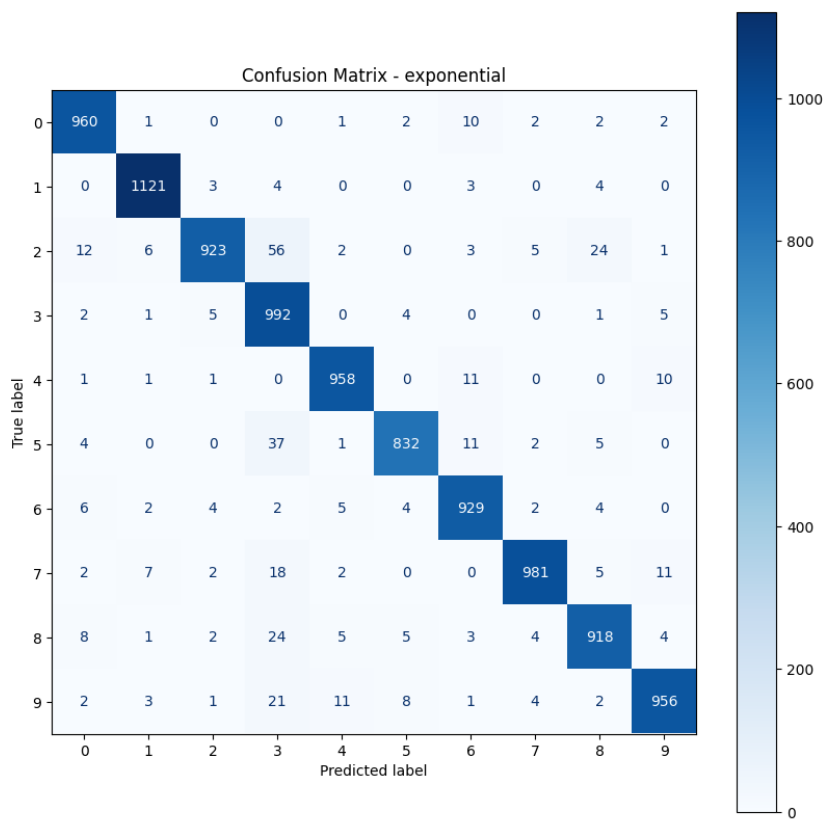
Confusion Matrixes

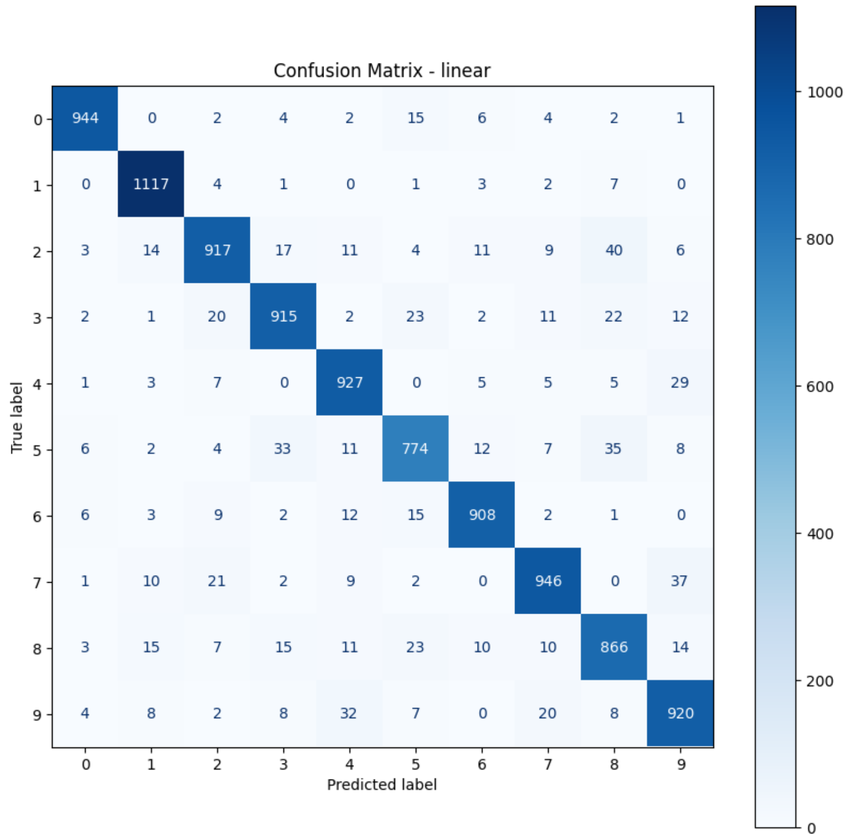
I have generated confusion matrixes for each activation function's model to visualize the difference in the way the models classify given items. As presumed based on previous result linear and exponential activation function perform far worse than the others.


Confusion Matrix - leaky_relu
True label
Predicted label
969 0 1 2 1 2 2 1 1 1
0 1123 3 1 0 1 2 1 4 0
2 1 1006 9 2 0 1 6 5 0
0 0 5 989 0 4 1 5 3 3
0 0 8 2 942 2 4 3 2 19
3 0 0 12 1 867 5 1 2 1
6 3 2 1 5 3 936 1 1 0
1 5 10 5 1 1 0 996 1 8
3 0 9 20 2 11 2 6 917 4
2 4 0 8 7 1 1 5 1 980


Confusion Matrix - selu
True label
Predicted label
968 0 1 2 1 1 2 2 1 2
0 1124 2 1 0 1 4 1 2 0
6 3 995 6 3 0 3 6 9 1
0 1 5 991 1 0 0 6 0 6
0 0 5 0 965 0 1 4 1 6
4 1 0 12 5 851 6 2 5 6
5 3 0 1 8 4 937 0 0 0
0 4 7 3 1 0 0 1008 0 5
4 0 4 5 5 5 4 8 933 6
3 4 1 5 14 2 0 11 0 969


Confusion Matrix - elu
True label
Predicted label
0 1 2 3 4 5 6 7 8 9
966 0 1 0 0 4 5 1 2 1
0 1123 3 1 0 1 3 1 3 0
5 2 998 4 2 0 5 6 9 1
1 0 4 981 0 8 1 7 5 3
0 0 1 0 962 1 6 1 1 10
4 1 0 7 2 868 5 1 2 2
3 3 0 1 1 4 942 1 3 0
0 7 9 4 2 0 0 996 2 8
2 1 2 9 6 6 4 4 937 3
3 7 2 6 10 2 2 5 7 965
1000
800
600
400
200
0


Confusion Matrix - exponential
True label
Predicted label
0 1 2 3 4 5 6 7 8 9
960 1 0 0 1 2 10 2 2 2
0 1121 3 4 0 0 3 0 4 0
12 6 923 56 2 0 3 5 24 1
2 1 5 992 0 4 0 0 1 5
1 1 1 0 958 0 11 0 0 10
4 0 0 37 1 832 11 2 5 0
6 2 4 2 5 4 929 2 4 0
2 7 2 18 2 0 0 981 5 11
8 1 2 24 5 5 3 4 918 4
2 3 1 21 11 8 1 4 2 956
1000
800
600
400
200
0



Summary

Activation Functions:

The choice of activation function significantly influences the performance and behavior of neural network models.

- **ReLU** (Rectified Linear Unit) emerged as one of the most effective activation functions, owing to its simplicity and ability to mitigate the vanishing gradient problem.
- **Tanh** (Hyperbolic Tangent) and Softmax showed competitive performance, particularly in tasks requiring outputs within specific ranges or for multi-class classification.
- **Sigmoid** activation function, while suitable for binary classification tasks, demonstrated limitations in deeper networks due to the vanishing gradient issue.
- **Leaky ReLU, SeLU, eLU**, and **Exponential** activation functions exhibited mixed results, with varying degrees of effectiveness depending on the specific dataset and task.

- **Linear activation function** generally performed poorly compared to other functions, likely due to its linearity and inability to introduce non-linearity into the network.

Batch Sizes:

- The choice of batch size plays a crucial role in the efficiency and effectiveness of model training.
- Smaller batch sizes, such as 8 or 16, facilitate faster convergence but may lead to increased computational overhead due to more frequent parameter updates.
- Larger batch sizes, such as 256 or 512, provide more stable updates and better utilization of computational resources but may require more epochs to converge and could suffer from overfitting if the batch size is too large relative to the dataset size.
- Batch sizes ranging from 32 to 128 generally exhibited favorable performance, striking a balance between computational efficiency and model convergence.
- The optimal batch size may vary depending on the dataset size, complexity, and available computational resources. It is crucial to experiment and tune the batch size based on these factors to achieve optimal performance.

In conclusion, selecting appropriate activation functions and batch sizes is critical in deep learning model development. It involves a trade-off between computational efficiency, convergence speed, and model performance. Experimentation and empirical testing are essential to identify the most suitable configurations for specific tasks and datasets. Additionally, considering other factors such as learning rate and regularization techniques can further enhance model performance and generalization ability.

Sidenotes

- from tensorflow.keras.utils import to_categorical # Numerical labels for texts

- from tensorflow.keras.models import Sequential # no cycles - signal goes all the way through

- from tensorflow.keras.layers import Dense
  # All full mesh - connection between neurons to all neurons between layers

- from tensorflow.keras.optimizers import SGD
  # optimizer - stochastic gradient descend